

软件测试

教学做一体化教程

朱毅 王淑华 陈恒 王雅轩◎编著

清华大学出版社

软件测试教学做一体化教程

朱 毅 陈 恒 王雅轩 编著

清华大学出版社
北 京

内 容 简 介

本书采用“教、学、做”一体化的方式撰写，合理地组织学习单元，并将每个单元分解为核心知识、能力目标、任务驱动、实践环节4个模块。全书共8章，第1章是软件测试基本概念，第2章是软件测试基本技术和测试过程，第3、4章是测试方法中的黑盒测试和白盒测试，第5章是测试管理以及自动化测试，第6～8章是软件测试工具QTP、LoadRunner、JUnit的理论与实践。书中实例侧重实用性和启发性、趣味性强、分布合理、通俗易懂，使读者能够快速掌握软件测试的基础知识、测试管理技术以及软件测试工具的使用技巧，为适应实战应用打下坚实的基础。

本书适合作为高等院校“软件工程”课程的教材或教学参考书，也适合作为有一定经验的软件工作人员的参考用书。

图书在版编目(CIP)数据

软件测试教学做一体化教程/朱毅，陈恒，王雅轩编著. --北京：清华大学出版社，2014
ISBN 978-7-302-36593-8

Ⅰ. ①软…　Ⅱ. ①朱…　②陈…　③王…　Ⅲ. ①软件－测试－高等学校－教材　Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2014)第112112号

责任编辑：田在儒
封面设计：王跃宇
责任校对：李　梅
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795764
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：11.5　　**字　　数**：260千字
版　　次：2014年6月第1版　　**印　　次**：2014年6月第1次印刷
印　　数：1～　000
定　　价：　.00元

产品编号：045664-01



本书按照“教、学、做”一体化模式精编了软件测试的核心内容，以核心知识、能力目标、任务驱动和实践环节为单元组织本书的体系结构。核心知识体现最重要和实用的知识，是教师需要重点讲解的内容；能力目标提出学习核心知识后应具备的能力；任务驱动给出了教师和学生共同来完成的任务；实践环节给出了需要学生独立完成的实践活动。

全书共8章。第1章概括地介绍了软件测试的基本概念，包括软件测试产生的背景、软件测试基础理论以及目前软件测试的岗位需求。第2章介绍了软件测试基本技术和测试过程，包括软件测试的分类方法、测试过程中的模型、软件测试阶段和开发阶段的对照关系、测试工作流程以及测试计划和测试用例的编写。第3章介绍了黑盒测试的基本原理以及主要的黑盒测试方法，包括等价类划分法、边界值分析法、决策表分析法，并结合案例进行方法讲解以及测试用例的设计。第4章介绍了白盒测试的基本原理以及主要的白盒测试方法，包括逻辑覆盖法、独立路径测试法、面向对象的白盒测试，并结合案例进行方法讲解以及测试用例的设计。第5章介绍了软件测试管理及自动化测试的基本概念，包括软件测试管理计划、软件测试管理过程、软件缺陷管理的定义以及缺陷的属性、软件测试管理周期以及自动化测试发展的必然性。第6章介绍了功能测试工具QTP的基本工作原理以及实际应用，包括QTP工作流程、检查点设置、参数化测试脚本、输出值类型等。第7章介绍了负载测试工具LoadRunner的基本工作原理以及实际应用，包括核心组件的使用、制订测试计划、创建测试脚本、定义场景、执行场景和结果分析等。第8章介绍了单元测试工具JUnit的基本工作原理以及实际应用，包括JUnit测试框架中的核心类、使用断言方法来实现单元测试等。

本书特别注重引导学生参与课堂教学活动，适合高等院校相关专业作为“教、学、做”一体化的教材。

编　者

2014年1月


目 录
CONTENTS

第 1 章

软件测试概述

主要内容

- 软件测试背景
- 软件测试基础理论
- 软件测试的岗位需求

在学习软件测试之前，应该具有一定的计算机编程基础、数据库管理基础以及软件工程等理论基础。学习软件测试之后，可以采用相关测试理论以及测试方法、测试工具，有针对性地对已有软件产品或者待开发的软件产品进行相关的测试工作，保证软件的质量，提高客户的满意度。在本章中主要掌握软件测试背景、软件测试的发展现状、软件测试的基础理论、软件测试的岗位需求等相关内容。

1.1 软件测试背景

1.1.1 核心知识

随着计算机与信息技术的飞速发展，软件产品已经应用到社会的各个领域之中，与人们现实中的生活息息相关，软件产品的质量自然成为人们共同关注的焦点。在软件行业激烈的竞争环境中，软件开发商为了占有市场，避免被淘汰，必须重视软件产品的质量。用户为了保证自己业务能顺利完成，也希望能够选用优质的软件。质量不佳的软件产品不仅会使开发商的维护费用和用户的使用成本大幅增加，还可能产生其他的责任风险，造成公司信誉下降。在一些关键应用（如银行系统、证券交易系统、铁路调度系统、民航订票系统、军事防御和核电站安全控制系统等）中使用质量有问题的软件，还可能造成灾难性的后果。

软件危机曾经是软件界甚至整个计算机界最热门的话题。为了解决这场危机，软件从业人员、专家和学者做出了大量的努力。现在人们已经逐步认识到所谓的软件危机实际上仅是一种状况，那就是软件中有错误，正是这些错误导致了软件开发在成本、进度和质量上的失控。软件中存在错误是不可避免的，因为软件是由人来完成的，所有由人做的工作都不会是完美无缺的。问题在于软件开发人员如何去避免错误的产生和消除已经产生的错误，使程序中的错误密度达到尽可能低的程度。

软件测试作为软件产品质量保障的重要手段之一，越来越受到软件开发商的重视，为了

保证软件质量,所以软件产品必须要进行软件测试。软件测试已经成为软件开发中必不可少的环节。统计表明,在典型的软件开发项目中,软件测试工作量往往占软件开发总工作量的40%以上,而在软件开发的总成本中,用在测试上的开销要占30%~50%。如果把维护阶段也考虑在内,讨论整个软件生存期时,测试的成本比例也许会有所降低,但实际上维护工作相当于二次开发,乃至多次开发,其中必定还包含有许多测试工作,软件开发在资金上的平均投入如图1.1所示。

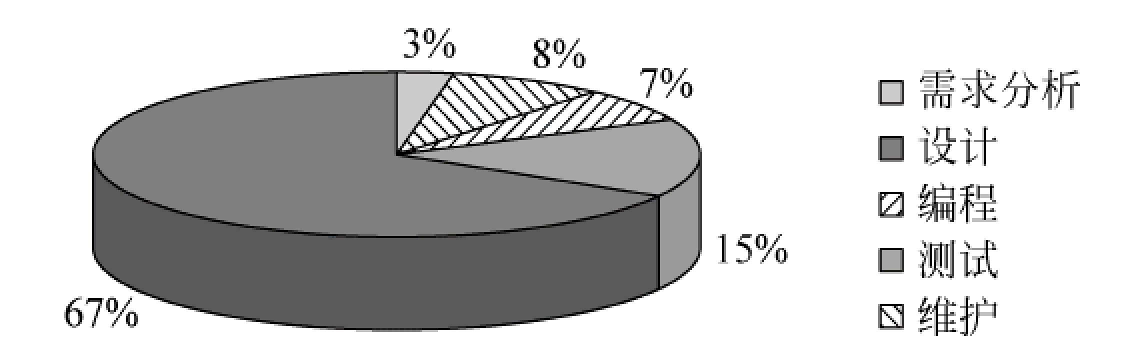


图1.1　软件开发在资金上的平均投入

从图中可以看出,测试和维护的成本之和占到软件开发总成本的80%左右。因此,测试对于软件产品来说是必需的,问题在于应该考虑采用什么方法、如何进行软件测试。

1. 软件

在测试之前,首先了解一下什么是软件。

直观上说,当人们提起软件时,通常想到的是软件产品。如Word、个人理财管家、腾讯QQ等,用户知道如何操作软件就可以。稍微有点计算机基础知识的用户,能更深一步地知道,软件是通过特定的编程语言和开发工具,组合在一起形成的程序或软件包。这种程序或者软件包通常称之为“代码”。这样的理解也多少有些片面。

权威机构IEEE(电气电子工程师学会,世界上最大的专业技术组织之一)给出了软件的定义:软件是由计算机程序、规程以及可能的相关文档和运行计算机系统所需要的数据组成。即,软件=程序+规程+文档+数据。

程序:也称为“代码”。用来激活计算机执行所需的应用。

规程:用来确定执行程序的顺序和进度安排,应用的方法。

文件:针对开发人员、用户和维护人员有不同类型的文档。开发文档(需求分析报告、设计报告、程序注释文档等)使得开发组成员能够进行高效的合作,能够对软件设计与最终产品进行有效的评审和审查。用户文档(用户手册)使用户知道如何使用软件产品。维护文档(程序员软件手册)向维护组提供每个软件模块的结构和任务所需要的信息,在软件产品出现错误(Bug)时,能快速地定位错误起因以及对软件进行及时更新和维护。

数据:包括代码、参数、数据表等数据。另一种重要数据是标准测试数据,用来测试软件的功能、性能是否存在一定的问题。

2. 软件缺陷及原因

软件缺陷(Bug)是指计算机系统或者程序中存在的任何一种破坏其正常运行能力的问题、错误或者隐藏的功能缺陷、瑕疵。

一个Bug影响力到底有多大呢?下面的事实也许更具有说服力。

(1)“爱国者”导弹防御系统。美国爱国者导弹防御系统1991年首次被用在第一次海湾战争对抗伊拉克“飞毛腿”导弹的防御作战中,总体上看效果不错,赢得各界的赞誉,但它

还是有几次失利，没有成功拦截伊拉克“飞毛腿”导弹，其中一枚在沙特阿拉伯的多哈爆炸的“飞毛腿”导弹造成28名美国士兵死亡。分析专家发现，拦截失败的症结在于一个软件缺陷，当“爱国者”导弹防御系统的时钟累计运行超过14小时后，系统的跟踪系统就不准确了。在多哈袭击战中，“爱国者”导弹防御系统运行时间已经累计超过100多个小时，显然那时的跟踪系统已经很不准确，从而造成这种结果。

(2) 英特尔奔腾芯片缺陷。在计算机的“计算器”程序中输入以下算式：(4195835/3145727)×3145727－4195835，如果答案是0，就说明该计算机浮点运算没问题；如果答案不是0，就表示计算机的浮点除法存在缺陷。1994年，英特尔奔腾CPU芯片就曾经存在这样一个软件缺陷，而且被大批生产出来卖到用户那里。最后，英特尔为自己处理软件缺陷的行为道歉，并拿出4亿多美元来支付更换坏芯片的费用。

(3) “冲击波”计算机病毒。“冲击波”计算机病毒首先在美国发作，使美国的政府机关、企业及个人用户的成千上万计算机受到攻击。随后，冲击波蠕虫很快在因特网上广泛传播，中国、日本和欧洲等国家和地区也相继受到不断的攻击，结果造成十几万台邮件服务器瘫痪，给整个世界范围内的Internet通信带来惨重损失。后来调查发现，制造冲击波蠕虫的黑客仅仅用了3周时间就制造了这个恶毒的程序，“冲击波”计算机病毒仅仅是利用微软Messenger Service中的一个缺陷，即可以攻破计算机安全屏障，可使基于Windows操作系统的计算机崩溃。该缺陷几乎影响当前所有微软Windows系统，它甚至使安全专家产生更大的忧虑：独立的黑客们将很快找到利用该缺陷控制大部分计算机的方法。随后，微软公司不得不紧急发布补丁包，修正这个缺陷。

美国商务部下属的国力标准技术研究所(NIST)有关软件缺陷的调查报告显示：“由于软件缺陷引起的损失额每年高达595亿美元，这一数字相当于美国国民生产总值的0.6%。”

软件缺陷会造成软件质量的下降，最终导致软件产品在某种程度上不能满足用户的需求。软件缺陷可以是代码缺陷、规程缺陷、文件缺陷或者软件数据缺陷等。所有引起这些缺陷的原因都是人，是由系统分析员、程序员、软件测试人员、文档专家、客户经理，甚至有的时候是由客户和他们的代表造成的。

软件缺陷存在于软件产品的整个生存周期中，通常是由以下9个方面引起的。

(1) 需求定义不完全。软件需求定义不完全是产生软件缺陷的最主要原因之一。此类最常见的问题有：需求的错误性定义；缺少一些重要的需求；需求定义的不完备；包含了不必要的需求等。

(2) 客户与开发人员沟通不畅。客户与开发人员沟通不畅是开发过程早期阶段出现的主要问题。此类最常见的问题有：系统分析员没有针对不确定的需求和客户进行反复确认；客户对需求表述不清；客户的需求变更没有落实在书面上；客户对开发者提出的设计问题存在误解等。

(3) 开发人员对软件需求的偏离。在一些情况下，软件开发者可能偏离文档化的需求，这种行为经常引起软件缺陷。常见的情况有：开发者重用以前项目中的模块，却没有对用户新需求进行更改和进行适当的分析；由于时间或者经费问题，开发者为应对开发压力删去部分需求的功能；开发者未得到客户的批准，对软件进行改进等。

(4) 逻辑设计错误。当设计系统的专业人员(系统分析师、软件工程师、分析员等)系统

地阐述软件需求时，软件缺陷就可能进入系统。典型的错误有：通过错误的算法表达软件需求；过程前后顺序的错误；对系统边界条件定义的错误；漏掉需要的软件系统状态；漏掉定义软件系统非法操作的反应等。

（5）编码错误。各种各样的原因可以使程序员产生编码错误。包括误解设计文档；编程语言中的语言性错误；开发工具的应用错误；数据选择错误等。

（6）不符合文档编写规范。几乎所有软件公司都有自己的文档编制与编码规范，这些标准定义了文档的内容、次序和格式以及程序员编写的代码。为了支持这种要求，软件公司会公布文档模板和编码规范，要求所有开发组和成员必须遵守这些要求。不遵守文档规范，会使软件的后期处理增加出错的几率。

（7）测试过程不足。测试过程的不足会留下大量未检测到和未改正的错误，从而影响出错率。这些不足主要来自于：不完备的测试计划；检测到的错误和缺陷未记入文档和报告；由于没有合适地指出缺陷原因，未能及时改正检测到的软件缺陷等。

（8）规程错误。规程向用户指引每个处理步骤所需的活动。它们对复杂软件系统尤其重要，因为处理是分若干个步骤进行的，每一步都要输入各种类型的数据并允许检查中间结果。

（9）文档编制错误。给开发和维护组制造麻烦的通常都是文档编制错误，在设计文档中的错误会最终体现在软件产品中。另一类文档编制错误主要影响用户，如在用户手册和软件"帮助"中显示的错误，会造成用户的不正确地使用软件产品。

软件缺陷按照对用户使用影响的严重级别可以分为以下几种。

（1）建议型：通常为可用性方面的一些建议，如字体颜色等一些不影响使用的问题。

（2）提示型：软件产品中存在一些小问题，如有个别错别字、文字排版不整齐等，对功能几乎没有影响，软件产品仍可使用。

（3）一般型：软件产品中存在不太严重的错误，如次要功能模块丧失、提示信息不够准确、用户界面差和操作时间长等。

（4）严重型：软件产品中存在严重错误，指功能模块或特性没有实现，主要功能部分丧失，次要功能全部丧失或致命的错误声明。

（5）致命型：软件产品中存在致命的错误，造成系统崩溃、死机或造成数据丢失、主要功能完全丧失等。

3. 软件质量

软件测试的目的是保证软件产品的质量。在分析了软件缺陷的原因后，那什么是软件质量呢？

IEEE 指出：软件质量是系统、部件或过程满足规定需求的程度，是满足顾客或用户需求的程度。

软件质量本质上就是软件产品符合用户的需求，达到用户的满意度。影响软件质量的主要因素从管理角度可划分为以下三组。

（1）从产品运行上可以分为：正确性、健壮性、效率、完整性、可用性、风险；

（2）从产品修改上可以分为：可理解性、可维修性、灵活性、可测试性；

（3）从产品转移上可以分为：可移植性、可再用性、互运行性。

考量软件质量主要有以下 11 个指标。

(1) 性能(Performance)是指系统的响应能力，即要经过多长时间才能对某个事件作出响应，或者在某段时间内系统所能处理的事件个数；

(2) 可用性(Availability)是指系统能够正常运行的时间比例；

(3) 可靠性(Reliability)是指系统在应用或者错误面前，在意外或者错误使用的情况下维持软件系统功能特性的能力；

(4) 健壮性(Robustness)是指在处理或者环境中系统能够承受的压力或者变更能力；

(5) 安全性(Security)是指系统向合法用户提供服务的同时能够阻止非授权用户使用的企图或者拒绝服务的能力；

(6) 可修改性(Modification)是指能够快速地以较高的性能价格比对系统进行变更的能力；

(7) 可变性(Changeability)是指体系结构扩充或者变更成为新体系结构的能力；

(8) 易用性(Usability)是衡量用户使用软件产品完成指定任务的难易程度；

(9) 可测试性(Testability)是指软件发现故障并隔离定位其故障的能力特性，以及在一定的时间或者成本前提下进行测试设计、测试执行的能力；

(10) 功能性(Function ability)是指系统能完成所期望工作的能力；

(11) 互操作性(Inter-Operation)是指系统与外界或系统与系统之间的相互作用能力。

4. 软件质量保证

软件质量保证(Software Quality Assurance，SQA)即参照一定的质量标准、目标及各项软件流程、规范来监督、管理公司软件产品的质量；它的目的是为了客观地核实软件项目的实施行动与开发中的产品是否遵从于对应的需求、过程描述、标准及规程。

软件质量保证的目标主要包括以下 4 个方面：

(1) 通过监控软件开发过程来保证产品质量；

(2) 保证开发出来的软件和软件开发过程符合相应标准与规程；

(3) 保证软件产品、软件过程中存在的不合理问题得到处理，必要时将问题反映给高级管理者；

(4) 确保项目组制订的计划、标准和规程适合项目组需要，同时满足评审和审计需要。

SQA 与两种不同的参与者相关：一种是做技术工作的软件工程师，另一种是负责质量保证的计划、监督、记录、分析及报告工作的 SQA 小组。

软件工程师通过采用可靠的技术方法和措施，进行正式的技术评审，执行计划周密的软件测试来考虑质量问题，并完成软件质量保证和质量控制活动。

SQA 小组的职责是辅助软件工程小组得到高质量的最终产品。SQA 小组主要完成以下 6 项工作。

(1) 为项目准备 SQA 计划。该计划在制定项目规定、项目计划时确定，由所有感兴趣的相关部门评审。

(2) 参与开发项目的软件过程描述。评审过程描述以保证该过程与组织政策、内部软件标准、外界标准以及项目计划的其他部分相符。

(3) 评审各项软件工程活动，对其是否符合定义好的软件过程进行核实，记录、跟踪与过程的偏差。

(4) 审计指定的软件工作产品,对其是否符合事先定义好的需求进行核实。对产品进行评审,识别、记录和跟踪出现的偏差;对是否已经改正进行核实;定期将工作结果向项目管理者报告。

(5) 确保软件工作及产品中的偏差已记录在案,并根据预定的规程进行处理。

(6) 记录所有不符合的部分并报告给高级领导者。

1.1.2 能力目标

掌握软件缺陷的定义以及产生原因;掌握软件质量的定义及考核指标;掌握软件质量保证的定义及软件质量保证的工作流程。

1.1.3 任务驱动

1. 分组讨论什么是好的软件产品。

2. 关于软件测试对软件质量的意义,有以下观点:①度量与评估软件的质量;②保证软件质量;③改进软件开发过程;④发现软件错误。你认为哪些观点是正确的?并说明原因。

1.1.4 实践环节

1. 通过网络搜索曾经发生软件缺陷的重要事件并分析其产生的原因。

2. 接触一种软件产品,写出该软件产品的主要功能以及使用感受,并分析该软件可能存在的缺陷,在哪些方面可以进行改进。

1.2 软件测试基础理论

1.2.1 核心知识

1. 软件测试的定义

软件测试是通过人工或者自动化手段来运行或测试某个系统的过程,它是验证软件是否能够达到客户期望功能的唯一有效方法,也是保证软件产品质量的唯一途径。软件测试并非是简单的“挑错”,而是贯穿于软件生产过程的始终,是一套完善的质量体系。这要求测试工程师具备系统的测试专业知识及对软件的整体把握能力。

软件测试概念的确定,曾经经历过以下两种方法的争论。

(1) 20 世纪 70 年代初期,Bill. Hetzel 提出“评价一个程序和系统的特性或能力,并确定它是否达到预期的结果。软件测试就是以此为目的的任何行为。”他的核心观点是:测试是验证软件是“工作的”,以正向思维,针对软件系统的所有功能点,逐个验证其正确性。这就是软件测试的第一类方法。

(2) Glenford. Myers 提出“测试不应着眼于验证软件是工作的,应该认定软件是有错误的,然后用逆向思维去发现尽可能多的错误。测试是为发现错误而执行的一个程序或者系统的过程。”他的核心观点是:测试是为了证明程序有错,而不是证明程序无错误;一个好的测试用例是在于它能发现至今未发现的错误;一个成功的测试是发现了至今未发现的错

误的测试。这是软件测试的第二类方法。

目前广泛采用的是第二类方法。

2. 软件测试的目的

基于不同的立场,存在着两种完全不同的测试目的。从用户的角度出发,普遍希望通过软件测试暴露出软件中隐藏的错误和缺陷,以考虑是否可以接受该产品。而从软件开发者的角度出发,则希望测试成为表明软件产品中不存在错误的过程,验证该软件已正确地实现了用户的要求,确立用户对软件质量的信心。

因为在程序中往往存在着许多预料不到的问题,可能会被疏漏,许多隐藏的错误只有在特定的环境下才可能暴露出来。如果不把着眼点放在尽可能查找错误这样一个基础上,这些隐藏的错误和缺陷就查不出来,会遗留到运行阶段中去。如果站在用户的角度替他们设想,就应当把测试活动的目标对准揭露程序中存在的错误。在选取测试用例时,主要考虑那些易于发现程序错误的数据。

下面这些观点可以看作是测试的目的或定义:

(1) 测试是为了发现程序中的错误而执行程序的过程;

(2) 好的测试方案是发现迄今为止尚未发现的错误的测试方案;

(3) 成功的测试是发现了至今为止尚未发现的错误的测试。

从上述规则可以看出,测试的正确定义是"为了发现程序中的错误而执行程序的过程"。这和某些人通常想象的"测试是为了表明程序是正确的","成功的测试是没有发现错误的测试"等观点是完全相反的。正确认识测试的目的是十分重要的,测试目的决定了测试方案的设计。如果为了表明程序是正确的而进行测试,就会设计一些不易暴露错误的测试方案;相反,如果测试是为了发现程序中的错误,就会力求设计出最能暴露错误的测试方案。

由于测试的目的是暴露程序中的错误,从心理学角度看,由程序的编写者自己进行测试是不恰当的。因此,在集成测试阶段通常由其他人员组成测试小组来完成测试工作。此外,应该认识到测试绝不能证明程序是正确的。即使经过了最严格的测试之后,仍然可能还有没被发现的错误潜藏在程序中。测试只能查找出程序中的错误,不能证明程序中没有错误。

3. 软件测试的原则

在软件测试的过程中,通常应该遵循以下7个原则。

(1) 所有的测试都应追溯到用户需求。这是因为软件的目的是使用户完成预定的任务,满足其需求;而软件测试揭示软件的缺陷和错误,一旦修正这些错误就能更好地满足用户需求。

(2) 应尽早地和不断地进行软件测试。由于软件的复杂性和抽象性,在软件生命周期各阶段都可能产生错误,所以不应把软件测试仅仅看作是软件开发的一个独立阶段,而应当把它贯穿到软件开发的各个阶段中去。在需求分析和设计阶段就应开始进行测试工作,编写相应的测试计划及测试设计文档,同时坚持在开发各阶段进行技术评审和验证,这样才能尽早发现和预防错误,杜绝某些缺陷和错误,提高软件质量。测试工作进行得越早,越有利于提高软件的质量,这是预防性测试的基本原则。

(3) 在有限的时间和资源下进行完全测试并找出软件所有的错误和缺陷是不可能的,软件测试不能无限进行下去,应适时终止。因为,测试输入量大、输出结果多、路径组合多,

用有限的资源来达到完全测试是不现实的。

(4) 测试只能证明软件存在错误而不能证明软件没有错误,测试无法显示潜在的错误和缺陷,继续进一步测试可能还会找到其他错误和缺陷。

(5) 充分关注测试中的集群现象。在测试的程序段中,若发现的错误数目比较多,则残存在该程序段中的错误数目也会比较多,因此应当花较多的时间和代价测试那些具有更多错误数目的程序模块。

(6) 程序员应避免检查自己的程序。考虑到人们的心理因素,自己揭露自己程序中的错误是件不愉快的事,自己不愿意否认自己的工作;此外,由于思维定式,自己难以发现自己的错误。因此,测试一般由独立的测试部门或第三方机构进行,这样测试相对比较客观。

(7) 尽量避免测试的随意性。软件测试是有组织、有计划、有步骤的活动,要严格按照测试计划进行,要避免测试的随意性。

4. 软件测试的经济性

人们通常认为,开发一个程序是困难的,测试一个程序则比较容易。这其实是误解。设计测试用例是一项细致并需要高度技巧的工作,稍有不慎就会顾此失彼,发生不应有的疏漏。无论采用何种测试方法,由于测试情况数量巨大,都不可能进行完备的测试。

所谓完备测试,就是让被测程序在一切可能的输入情况下全部执行一遍。通常也称这种测试为“穷举测试”。在实际测试中,穷举测试工作量太大,实践上行不通,这就注定了一切实际测试都是不彻底的。当然就不能够保证被测试程序中不存在遗留的错误。

软件测试的总目标是充分利用有限的人力和物力资源,高效率、高质量地完成测试。为了降低测试成本,选择测试用例时应注意遵守“经济性”的原则。

(1) 要根据程序的重要性和一旦发生故障将造成的经济损失来确定它的测试等级。

(2) 要认真研究测试策略,以便能使用尽可能少的测试用例,发现尽可能多的程序错误。掌握好测试量是至关重要的,一位有经验的软件开发管理人员在谈到软件测试时曾这样说过:“不充分的测试是愚蠢的,而过度的测试是一种罪孽。”测试不足意味着让用户承担隐藏错误带来的危险,过度测试则会浪费许多宝贵的资源。

测试是软件生存周期中费用消耗最大的环节。测试费用除了测试的直接消耗外,还包括其他的相关费用。测试量的多少主要受以下几个因素影响。

(1) 软件系统目标。软件系统目标的差别在很大程度上影响所需要进行的测试数量。那些可能产生严重后果的系统必须要进行更多的测试。如,一个操作系统软件的测试量应该比一个应用软件系统更大。一个安全性级别较高的系统应该比一个普通的软件系统要求有更多的测试。

(2) 潜在的用户数量。一个系统的潜在用户数量在很大程度上也会影响测试数量。如,一个支持百万人同时在线购物的系统会比一个只在办公室中运行的有几百个用户的系统需要更多的测试。如果这两个系统出现问题的话,前一个系统的经济影响肯定比后一个系统要大得多。除此以外,在处理错误的时候,所花的代价的差别也很大。如果在内部系统中发现了一个严重的错误,在处理错误的时候的费用就相对少一些;如果要处理一个遍布全世界的错误就需要花费相当大的财力和精力。

(3) 信息的价值。在考虑测试的必要性时,还需要将系统中所包含的信息的价值考虑

在内,一个支持许多家大银行或众多证券交易所的客户机/服务器系统中含有经济价值非常高的内容。很显然这一系统需要比一个支持便利店的系统要进行更多的测试。这两个系统的用户都希望得到高质量、无错误的系统,但是前一种系统的影响比后一种要大得多。因此应该从经济方面考虑,投入与其经济价值相对应的时间和金钱去进行测试。

(4) 开发机构。一个没有标准和缺少经验的开发机构很可能开发出充满错误的系统。在一个建立了标准化流程和有很多经验的开发机构开发出来的系统中的错误相对会少很多,因此,对于不同的开发机构来说,所需要的测试量也是截然不同的。

(5) 测试的时机。测试量会随时间的推移发生改变。在一个竞争激烈的市场里,争取时间是制胜的关键,开始可能不会在测试上花多少时间,但几年后如果市场分配格局已经建立起来了,那么产品的质量就变得更重要了,测试量就要加大。测试量应该针对合适的目标进行调整。

5. 软件测试终止的标准

软件测试的经济性决定了软件测试不可能无休止地进行下去,必须有其终止的标准。具体的测试终止标准要依据各个公司具体情况来制定,不能一概而论。通常测试终止满足以下几个原则。

(1) 基于"测试阶段"的原则

每个软件的测试一般都要经过单元测试、集成测试、系统测试这几个阶段,测试人员可以分别对单元测试、集成测试和系统测试制定详细的测试结束点。每个测试阶段符合结束标准后,再进行下一个阶段的测试。

例如:单元测试,通常要求测试结束点必须满足"核心代码100%经过Code Review"、"功能覆盖率达到100%"、"代码行覆盖率不低于80%"、"不存在A、B类缺陷"、"所有发现缺陷至少60%都纳入缺陷追踪系统且各级缺陷修复率达到标准"等标准。

(2) 基于"测试用例"的原则

测试设计人员设计测试用例,并请项目组成员参与评审,测试用例一旦评审通过,在后期测试时,就可以作为测试结束的一个参考标准。

例如:在测试过程中,如果发现测试用例通过率太低,可以拒绝继续测试,等待开发人员修复后再继续后期的测试。在功能测试用例通过率达到100%,非功能性测试用例达到95%以上时,允许正常结束测试。但是使用该原则作为测试结束点时,把握好测试用例的质量非常关键。

(3) 基于"缺陷收敛趋势"的原则

软件测试的生命周期中随着测试时间的推移,测试发现的缺陷图线,首先成逐渐上升趋势,然后测试到一定阶段,缺陷又呈下降趋势,直到发现的缺陷几乎为零或者很难发现缺陷为止。可以通过缺陷的趋势图线的走向,来决定测试是否可以结束,这也是一个判定标准。如图1.2所示,随着测试工作量的不断增加,缺陷数量呈下降趋势。

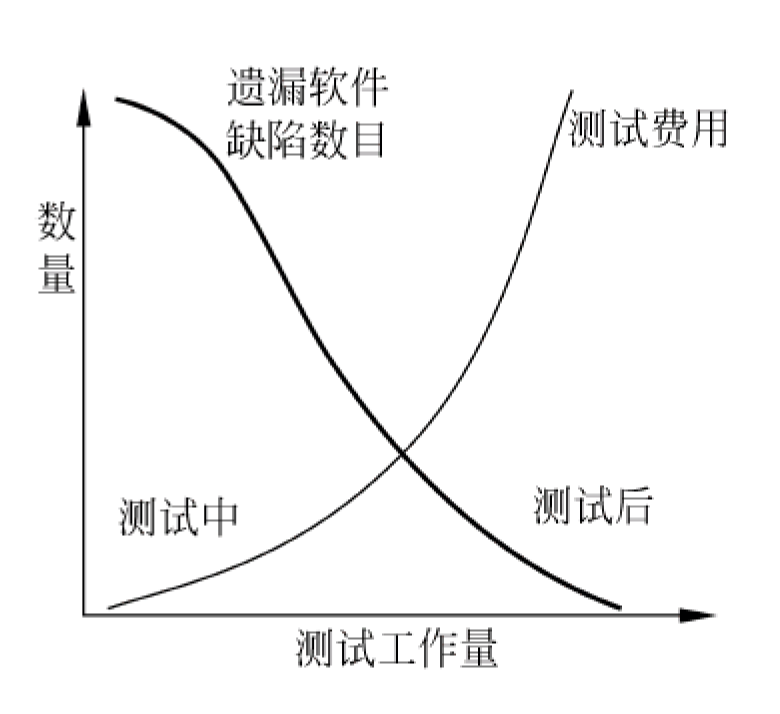


图1.2　软件缺陷趋势图

(4) 基于"缺陷修复率"的原则

软件缺陷在测试生命周期中可以分成几个严重等

级，分别是：严重错误、主要错误、次要错误、一般错误、较小错误和测试建议6种。测试人员在确定测试结束点时，必须保证严重错误和主要错误的缺陷修复率达到100%，不允许存在功能性的错误；次要错误和一般错误的缺陷修复率必须达到85%以上，允许存在少量功能缺陷，后面版本解决；对于较小错误的缺陷修复率最好达到60%～70%以上。对于测试建议的问题，可以暂时不用修改。

(5) 基于"验收测试"的原则

当软件测试进行到一定阶段后，达到或接近测试部门指定的标准时，就可以递交用户做验收测试。如果通过用户的测试验收，就可以立即终止测试。

1.2.2 能力目标

掌握软件测试的含义和测试目的；掌握软件测试的基本原则；了解软件测试的经济性；了解软件测试终止的常见标准。

1.2.3 任务驱动

1. 简述软件测试的意义。

2. 以下说法正确的有哪些？

(1) 因为测试工作简单，对软件产品影响不大，所以可以把测试作为新员工的一个过渡工作，或安排不合格的开发人员做测试。

(2) 测试是为了证明软件的正确性。

(3) 测试过程中应重视测试的执行，可以轻视测试的设计。

(4) 测试不能修复所有的软件故障。

(5) 测试工作是在交付软件产品的时候才进行的一项活动。

1.2.4 实践环节

1. 通过到软件企业实践调研，了解软件测试工作在软件产品生存周期中的比重和作用。

2. 对软件测试的经济性进行分析，进而分析"软件测试的资金投入不用过多"观点的正确性。

1.3 软件测试的岗位需求

1.3.1 核心知识

随着中国软件业的迅猛发展，软件产品的质量控制与质量管理正逐渐成为企业生存与发展的核心。为了保证软件在出厂时的"健康状态"，几乎所有的IT企业在软件产品发布前都需要大量的质量控制工作。作为软件质量控制中的重要一环，软件测试工程师应运而生。国内软件业因对软件质量控制的重要作用认识较晚，尚未形成系统化的软件测试人才需求供应链，造成了目前企业欲招纳软件测试人才，却"千金难求"的尴尬局面。

判断一个职业是否有前途需要以发展的眼光分析，既要看到短期的工资待遇，更要看到未来的发展空间；既要看到短期市场需求，更要看到长远的社会需求；既要看到职业的社

会地位，更要考虑到个人的职业兴趣。软件测试行业顺应全球化和信息化发展趋势，符合我国信息化与工业化发展目标，是新兴的朝阳职业。优秀的测试从业者依靠软件测试的专业技术，可以获得职业的不断提升，随着测试能力的提升，薪资待遇不断提升，成为受人尊敬的测试专家。

1. 软件测试职业规划

每个测试从业人员都希望通过努力，提高工作职位，实现个人价值。软件测试从业者有哪些岗位可以不断提高和发展呢？软件测试网的专家将软件测试职业进行全方位分析，提出了测试职业发展具有多级别、多层次、多方向、多职位的"四多"特征。软件测试人员职业发展的路线图如图1.3所示。

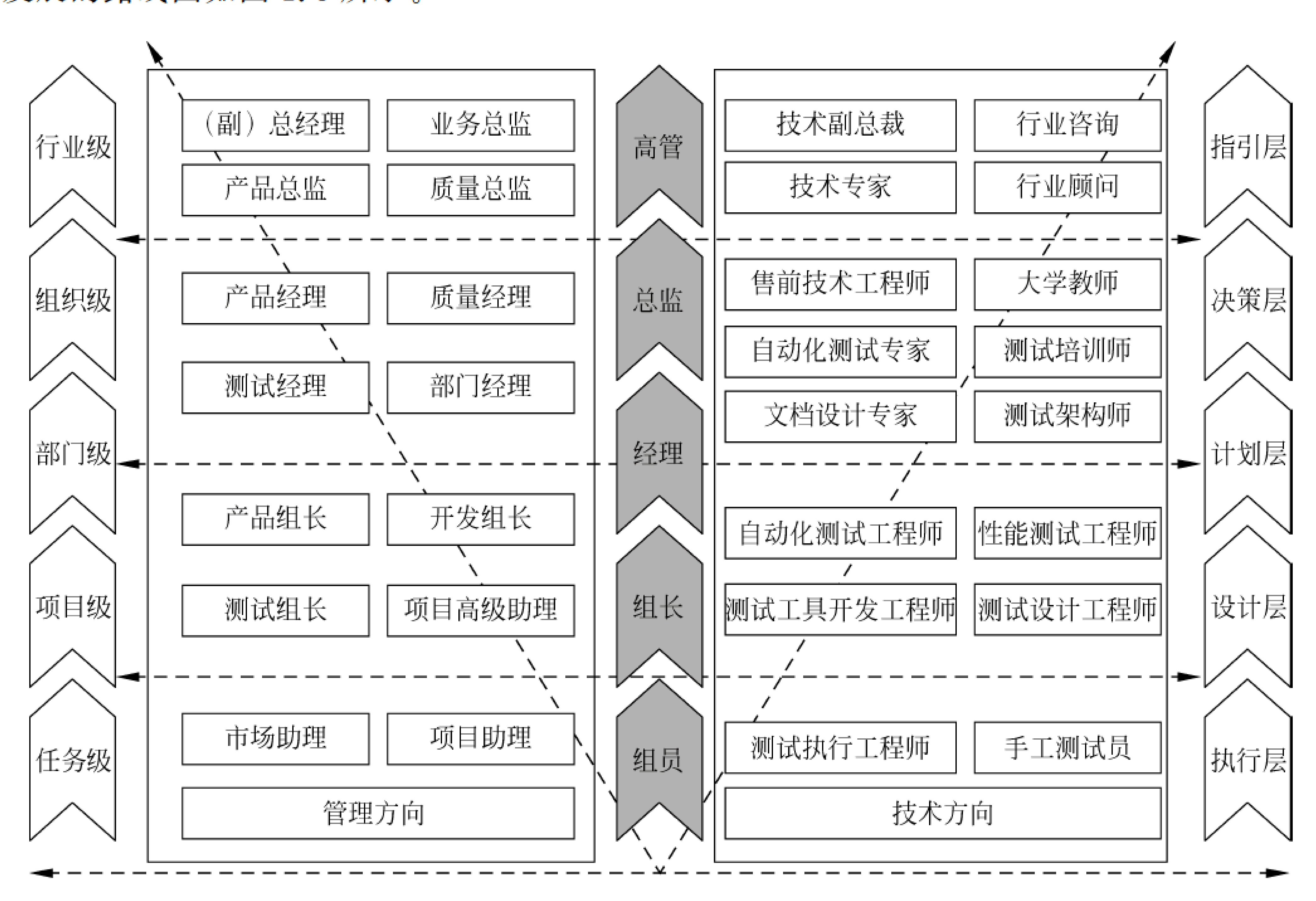


图1.3　软件测试职业发展规划图

（1）级别角度

描述了测试工作的影响范围，从小到大的各个级别分别是"任务级"、"项目级"、"部门级"、"组织级"和"行业级"。最小的测试工作影响范围只能影响到某个具体的测试任务，最高的测试工作可以影响到测试行业的发展趋势。

（2）层次角度

描述了测试工作在组织结构中的所在地位，从低到高的各个层次分别是"执行层"、"设计层"、"计划层"、"决策层"和"指引层"。测试工作最底层是软件测试的具体执行工作，最高层是测试工作可以指引测试行业的发展。

（3）方向角度

描述了测试工作的技能发展倾向，可以分为"技术"和"管理"两个方向。"技术"方向是在测试技术、领域技术和软件工程技术的广度和深度方面进行发展。"管理"方向是向提高

组织能力、领导能力、沟通协调方面深入发展。

(4) 职位角度

描述了测试工作对应的具体岗位类别是名称,职位类别可以分为“组员”、“组长”、“经理”、“总监”和“高管”,每个类别分别对应许多具体的测试岗位。目前大部分软件公司测试岗位与薪资待遇如表 1.1 所示。

表 1.1　测试岗位与薪资待遇表

测 试 岗 位	薪资待遇(元/月)	工作年限
初级测试工程师	2500~4500	1 年或 2 年
中级测试工程师	4500~8000	3 年或 4 年
高级测试工程师	8000~15000	5~8 年
资深测试工程师	>15000	>8 年

测试工作的职业发展方向决定测试职业的职位发展,测试职业发展的不同职业级别和层次影响测试职位的类别,不同的组织具有不同的测试职位名称及职责要求。软件测试强调实践性和应用性,无论今后向哪个方向发展,达到哪个级别和层次,最好从最基础的测试组员做起。

2. 测试工程师职业素质

一个测试工程师应该具备哪些职业素质,或者测试岗位有什么要求呢? 通常需要具备以下几种能力。

(1) 技术能力

作为一名测试工程师,不能仅仅从使用者的角度来测试软件产品,而且还要从技术的角度来设计测试用例,这里所说的技术包括基础的与专业的技术。

基础方面应学习的课程:编程语言、数据库理论、计算机网络技术、软件工程、数据结构、计算机组成原理等;

专业方面应掌握:软件测试基础、测试设计、自动化测试工具、软件质量管理、一门或多门外语等。

(2) 具有一定的编程经验

测试工程师需要对源码进行检查,特别是白盒测试工程师从程序结构的角度来测试软件,编写测试脚本,读懂源码对白盒测试人员来说是最基本的要求,而且如果有一定的编程经验的情况下,可以帮助测试人员对软件开发过程有较深入的理解,从编程人员的角度来正确地评价待测系统。

(3) 沟通能力

测试人员需要与很多人员进行沟通,项目经理、开发人员、客户、市场人员等都是测试人员经常需要沟通的对象,而且在面对不同的人员时,应该采用不同的语气、不同的态度,与客户要进行有效的沟通,处处为客户着想,客户就是上帝,与上帝说话要和颜悦色;与开发人员交往需要一定的技巧,开发人员开发出来的程序,测试人员需要“挑毛病”,双方在心理上经常处于一种敌对的态度,因此在说话的语气或讲述一个问题的出发点时特别要注意。

(4) 要有严谨、敢于承担责任、稳重的做事风格

思维严密,什么问题都要考虑到,当然除了做事认真仔细,也要有承担责任的勇气,在漫

长的项目实施过程中，或大或小的错误在所难免，要敢于承认错误。

（5）具有怀疑与破坏的精神

测试人员不能总是以常规的思路来测试软件，要设计一些非常规的、相反的测试用例来不断地“折磨”软件产品，要破坏性地测试，并且不要停止你的怀疑。

（6）善于自我总结、自我督促

应该说软件测试是一种既繁琐又枯燥无味的工作，做多了测试人员会觉得似乎一成不变，对自己的能力没有提高，这时候就需要作自我督促，并经常做一些阶段性的总结，新的技术、新的方法、新的工具层出不穷，要让自己跟上技术发展的脚步，善于将新技术、新方法、新工具应用到测试工作当中。

（7）团队合作

多参加团队活动，提高自己的团队作战能力。

1.3.2　能力目标

了解软件测试行业的发展趋势；了解软件测试岗位的需求；了解软件测试人员应该具备的职业素质。

1.3.3　任务驱动

1. 讨论软件测试行业发展的前景问题。
2. 如果你成为一名软件测试人员，你未来 5 年的职业规划是什么？

1.3.4　实践环节

1. 通过网络和软件公司调研，了解软件测试人员的招聘技术要求。
2. 针对软件公司的招聘要求，制定个人学习时间规划。

1.4　小　　结

- 软件测试是通过人工或者自动手段来运行或测试某个系统的过程，从而验证软件是否能达成期望功能，它是验证软件期望功能的唯一有效方法，也是保证软件产品质量的唯一途径。
- 软件测试的原则：所有的测试都应追溯到用户需求；应尽早地和不断地进行软件测试；在有限的时间和资源下进行完全测试找出软件所有的错误和缺陷是不可能的，软件测试不能无限进行下去，应适时终止；测试只能证明软件存在错误而不能证明软件没有错误；充分关注测试中的集群现象；程序员应避免检查自己的程序；尽量避免测试的随意性。
- 软件测试终止的标准：基于“测试阶段”的原则；基于“测试用例”的原则；基于“缺陷收敛趋势”的原则；基于“缺陷修复率”的原则；基于“验收测试”的原则。
- 测试工程师职业素质包括：技术能力；具有一定的编程经验；沟通能力；要有严谨、敢于承担责任、稳重的做事风格；具有怀疑与破坏的精神；善于自我总结、自我督促；团队合作精神。

习 题 1

1. 什么是软件测试？简述其目的与原则。
2. 影响软件测试的因素有哪些？
3. 简述软件测试终止的标准。
4. 软件测试人员应该具备哪些职业素质？
5. 在软件测试中，下面说法中错误的是（　　）。
   A. 测试是为了发现程序中的错误而执行程序的过程
   B. 测试是为了表明程序是正确的
   C. 好的测试方案是极可能发现迄今为止尚未发现的错误的方案
   D. 成功的测试是发现了至今为止尚未发现的错误的测试
6. 关于软件质量的描述，正确的是（　　）。
   A. 软件质量是指软件满足规定用户需求的能力
   B. 软件质量特性是指软件的功能性、可靠性、易用性、效率、可维护性、可移植性
   C. 软件质量保证过程就是软件测试过程
   D. 以上描述都不对
7. 为了提高软件测试的效率，应该（　　）。
   A. 随机地选取测试数据
   B. 取一切可能的输入数据作为测试数据
   C. 在完成编码以后制订软件的测试计划
   D. 选择发现错误可能性最大的数据作为测试用例
8. 软件测试的对象包括（　　）。
   A. 目标程序和相关文档
   B. 源程序、目标程序、数据及相关文档
   C. 目标程序、操作系统和平台软件
   D. 源程序和目标程序
9. 下面说法正确的是（　　）。
   A. 软件测试是一个贯穿软件开发生命周期的活动
   B. 软件测试只在编码后进行
   C. 测试过程中应重视测试的执行，可以轻视测试的设计
   D. 因为测试工作简单，对软件产品质量影响不大
10. 以下关于软件性能的说法中，正确的是（　　）。
    A. 软件性能与该软件的实现算法无关
    B. 软件的吞吐量越大，其平均响应时间总是越慢
    C. 给软件的可用资源越少，其平均响应时间越短
    D. 对于同一个网站，其支持的同时改善请求的用户数越大，该网站的性能越好

第 2 章

软件测试基本技术和测试过程

主要内容

- 软件测试分类
- 软件测试过程
- 软件测试工作流程

在本章中主要掌握软件测试的几种分类方法，如按测试阶段分类、按测试内部结构分类、按测试实施的组织以及按照程序是否运行分类四种方式；软件测试过程中的 V 模型、W 模型以及 H 模型；掌握软件测试几个阶段和开发阶段的对照关系；软件测试工作流程中的五个阶段，重点在于测试计划以及测试用例的编写。

2.1　软件测试分类

2.1.1　核心知识

软件测试在业界通常按照以下四种方式进行分类。

（1）按照开发阶段划分为：单元测试、集成测试、系统测试、确认测试、验收测试、回归测试。

① 单元测试。又称为模块测试，主要检查每个程序单元是否正确实现详细设计说明中的模块功能。

② 集成测试。又称为组装测试，将所有的程序模块进行有序、递增的测试，检验程序单元或部件之间的接口关系。

③ 系统测试。检查完整的程序系统能否和系统（包括硬件、外设和网络、系统软件、支持平台等）正确配置、连接，并满足用户需求。

④ 确认测试。用来证实软件是否满足特定用途的需求，是否满足软件需求说明书的规定。

⑤ 验收测试。按照项目任务或合同、供需双方签订的验收依据文档，进行整个系统的测试与评审，决定是否接受或拒收系统。

⑥ 回归测试。指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其他代码产生的错误。

(2) 按照测试方法是否针对系统内部结构与具体实现算法可以划分为黑盒测试、白盒测试和灰盒测试。

① 黑盒测试。也称为功能测试,它是通过测试来检测每个功能是否都能正常使用。着眼于程序外部结构,不考虑内部逻辑结构如何实现。主要针对软件界面和软件功能进行测试。

黑盒测试从产品功能角度测试,可以最大限度地满足用户的需求。相同的用户操作可以通过计算机模拟重复执行。依据测试用例有针对性地寻找问题,定位更为准确,容易生成测试数据。但是,黑盒测试中程序代码得不到测试,如果规格说明设计有误,错误将很难发现。测试结果的准确性取决于测试用例的设计。

② 白盒测试。也称为结构测试或逻辑驱动测试,它是按照程序内部的结构测试程序,通过测试来检测产品内部动作是否按照设计规格说明书的规定正常进行,检验程序中的每条通路是否都能按预定要求正确工作。这一方法是把测试对象看作一个打开的盒子,测试人员依据程序内部逻辑结构相关信息,设计或选择测试用例,对程序所有逻辑路径进行测试,通过在不同点检查程序的状态,确定实际的状态是否与预期的状态一致。

白盒测试迫使测试人员去仔细思考软件是如何实现的,可以检测代码中的每条分支和路径,揭示隐藏在代码中的错误,通常对代码的测试比较彻底。但是,白盒测试的代价比较昂贵,无法检到测试代码中遗漏的路径和数据敏感性错误,对系统功能性的掌握不够。

③ 灰盒测试。介于白盒测试与黑盒测试之间。可以这样理解,灰盒测试关注输出对于输入的正确性,同时也关注内部表现,但这种关注不像白盒那样详细、完整,只是通过一些表征性的现象、事件、标志来判断内部的运行状态,有时候输出是正确的,但内部其实已经错误了,这种情况非常多。如果每次都通过白盒测试来操作,效率会很低,因此需要采取这样的一种灰盒测试的方法。

相对于黑盒测试的优点:测试可以及早介入;有助于测试人员理解系统结构;有助于管理层了解真实的开发进度;可以构造更好的测试用例;利于提升测试人员能力。

相对于白盒测试的优点:无论从招聘和培训人员的角度考虑,人力资源的成本较低。

(3) 按照测试实施的组织划分可分为开发方测试、用户测试和第三方测试。

① 开发方测试。开发方通过检测和提供客观证据,证实软件的实现是否满足规定的需求,在开发环境下,开发方对提交的软件进行全面的自我检查。

② 用户测试。在用户的应用环境中,用户通过运行软件,检测软件实现是否符合自己预期的要求,这里指用户的使用性测试。

③ 第三方测试。介于软件开发方和用户方之间的测试组织的测试,通常测试结果较为客观。

(4) 按照被测软件是否运行可划分为静态测试和动态测试。

① 静态测试。不实际运行软件,发挥人的逻辑思维优势,主要对软件代码的逻辑、程序结构等方面进行评测。主要包括需求评审、设计评审、代码走查、代码检查等。

② 动态测试。实际运行软件,发现系统中存在的错误。单元测试、集成测试、系统测试、确认测试、验收测试、回归测试等各个测试阶段都包含有动态测试的内容。

常见的测试方法分类如表 2.1 所示。

表 2.1　常见测试方法分类表

方　　法	概　　述
文档测试	对相关的设计报告和用户使用说明进行测试
性能测试	也称压力或负载测试
健全性测试	常作为初始测试,确定一个新的软件版本是否表现正常
增量集成测试	增加新的功能后进行新的测试
兼容性测试	在一个特定的硬件、软件、操作系统、网络等环境下的性能状况
可用性测试	测试该软件的用户界面是否友好,使用是否简洁
安装或卸载测试	测试软件的安装、卸载或升级过程
恢复能力测试	测试系统在崩溃、硬件失效,或者遇到其他灾难性的问题时是否能很好地恢复
比较测试	在同类产品中比较软件的优缺点
α 测试	在软件开发将结束时进行该测试
β 测试	当开发和测试工作实质上完成时进行该类测试
安全性测试	验证系统安全性、保密性措施是否发挥作用,有无漏洞

2.1.2　能力目标

掌握常见软件测试分类的方法；掌握单元测试、集成测试、系统测试、确认测试、验收测试、回归测试的基本概念；掌握黑盒测试、白盒测试和灰盒测试的基本概念；掌握开发方测试、用户测试、第三方测试的基本概念；掌握静态测试、动态测试的基本概念。

2.1.3　任务驱动

1. 单元测试阶段是采用白盒测试还是黑盒测试,或者是两者结合,谁的比重较大些?
2. 集成测试阶段是采用白盒测试还是黑盒测试,或者是两者结合,谁的比重较大些?
3. 压力(负载)测试是什么? 属于哪个测试阶段?
4. 可用性测试指的是什么?
5. 为什么说第三方测试更客观些?

2.1.4　实践环节

1. 在软件测试人员招聘过程中,试分析白盒测试工程师的工作职责是什么。
2. 试分析在使用 Windows 操作系统时,由于故障突然死机,重启后系统自检的过程和软件测试中哪几个方法有关?
3. 你是否使用过 Beta 版本的软件产品,分析该软件产品属于软件周期中的什么阶段?

2.2　软件测试过程

2.2.1　核心知识

在整个测试过程中也满足一定的规律和模型。常见的软件测试过程模型有 3 种：V 模型、W 模型和 H 模型。

1. 软件测试过程 V 模型

V 模型是比较常见的软件测试模型图，模型图如图 2.1 所示。

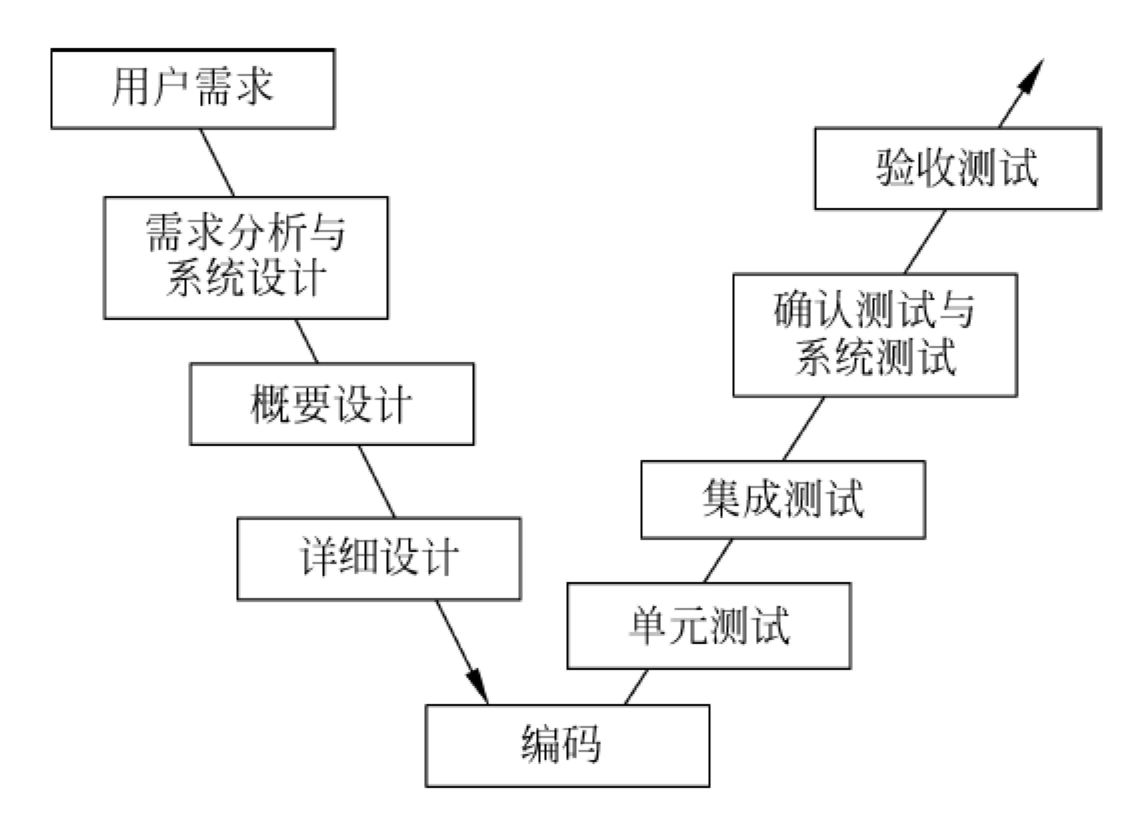


图 2.1　软件测试 V 模型图

V 模型是瀑布模型的一种变形，反映了软件测试活动与软件系统分析和设计的关系，明确地标明了测试过程中存在的不同级别，并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系。

在 V 模型中把测试阶段分为单元测试、集成测试、系统测试和验收测试。

1）单元测试

单元测试(Unit Testing)与详细设计阶段对应。又称单体测试、模块测试，是最小单位的测试。

什么是单元？

在面向过程的开发语言(如 C 语言)中，一个测试单元可以是一个函数、结构体、几个函数的集合等；在面向对象的开发语言(如 Java 语言)中，可以是一个方法、一个类，或者几个类的集合等。

单元测试的依据是详细设计、程序源代码和编码标准，对模块内所有重要的控制路径设计测试用例，以便发现模块内部的错误。通常单元测试需要由 4 个部分组成：详细设计说明书、程序代码清单、桩模块和驱动模块，它们之间的关系如图 2.2 所示。

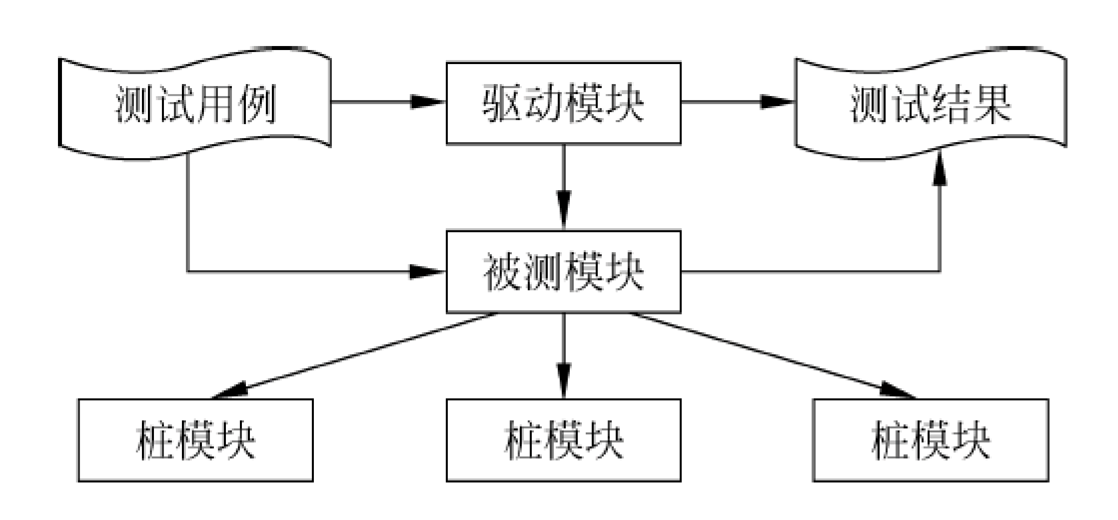


图 2.2　单元测试关系图

- 测试用例。根据详细说明书来设计测试用例。包括输入的数据、程序的流程、过程之间的调用，以及程序运行的结果等。
- 被测模块。待测的方法或者类。

- 驱动模块。用来调用被测模块的模块。如 Java 语言，在类中写好一个方法后，为了验证这个方法的有效性，可以在主方法 main 中去调用该被测方法，main 方法可以作为驱动模块出现。
- 桩模块。用来支撑被测模块的模块。如在网上银行系统中，想要测试转账模块是否好用，转账模块需要用户首先正确登录后才能使用，这时候登录模块就是转账模块的桩模块。

应用举例

假设某系统的程序结构图如图 2.3 所示。

说明：A 是顶级模块，包括 B、C、D 三个子模块，B 模块又由 E、F 两个子模块支撑，D 由一个 G 子模块支撑，模块间的调用关系满足从上层调用下层的关系，即 A 模块负责调用 B、C、D 模块，B 模块调用 E、F 模块，D 模块调用 G 模块。

假设系统中的各个模块交给同一项目组的不同程序员同时进行开发，如果要对 B 模块进行单元测试的话，而碰巧它的调用模块 A 以及它的子模块 E、F 又没有开发出来时，如何来进行单元测试呢？

这时候负责 B 模块单元测试的人员就应该按照详细说明书中的调用关系，模拟出整个 B 模块的流程关系，才可以对 B 模块进行单元测试。即使用驱动模块 a 代替 A 来调用 B 模块，使用桩模块 e、f 分别代替 E、F 对 B 模块进行支撑，如图 2.4 所示。

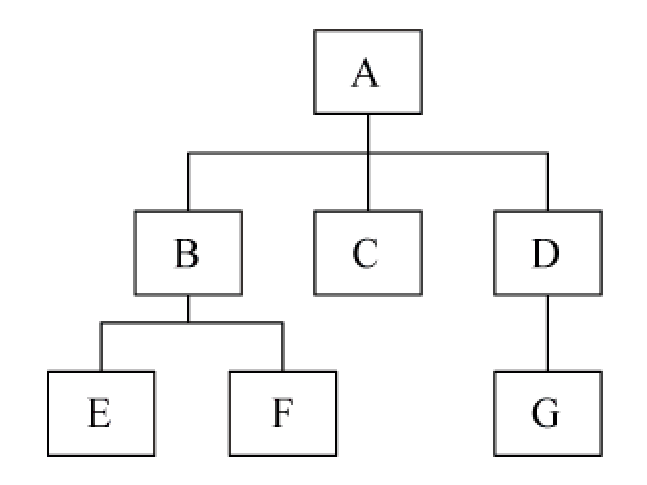


图 2.3　某系统程序结构图

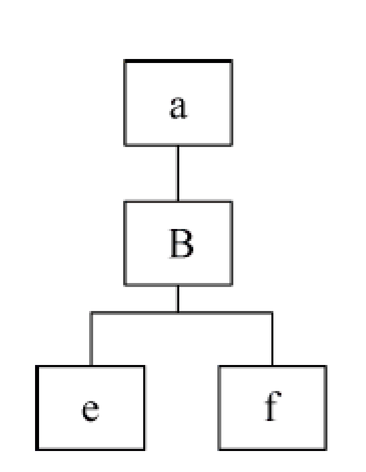


图 2.4　B 模块单元测试图

单元测试是所有的测试活动中最早进行的，它能以最低的成本发现和修复软件单元中的错误。单元测试使用白盒测试和黑盒测试方法对被测单元进行测试，以白盒方法为主。多个被测模块之间的单元测试可同时进行，以提高单元测试效率。绝大部分情况下，由白盒测试工程师做单元测试的设计和执行，如果由于公司规模较小的话，也可以由开发人员进行单元测试，但最好避免本模块的开发人员测试本模块，尽量进行同一项目组不同人员或者不同项目组之间的交叉测试。

2）集成测试

集成测试与概要设计阶段对应，又称组装测试。集成测试是在单元测试的基础上，将所有模块按照设计要求组装成子系统或系统进行的测试活动。

集成测试主要关注的问题有：模块之间接口的数据是否正确；模块之间会否产生不利的影响；各个子系统组合起来，能否达到预期要求的父功能；全局数据结构是否有问题。

单元测试与集成测试之间的区别如表 2.2 所示。

表 2.2 单元测试与集成测试的区别

比较内容 \ 测试类别	单 元 测 试	集 成 测 试
测试对象	实现具体功能的单元模块	模块组合
测试方法	白盒测试为主	黑盒测试为主
测试时间	最早	晚于单元测试
测试内容	模块内部程序的逻辑	验证各个接口、接口之间的数据传递关系、模块组合后能否达到预期效果

集成测试的策略主要分为两种：非增量式集成测试和增量式集成测试。

(1) 非增量式集成测试

非增量式集成测试(大爆炸式集成测试)是采用一步到位的方法来构造测试。对所有模块进行单元测试后，按照程序结构图将各模块连接起来，把连接后的程序当作一个整体进行测试。

应用举例

为了更好地描述集成测试模型，这里使用测试模型：

```
boolean Test(模块名);
```

表示对调用 Test 方法某个模块进行单元测试，测试通过返回 true，测试失败返回 false。

以上例中的系统举例。在进行集成测试前，已经对所有模块进行过单元测试。

即 Test(A)；Test(B)；Test(C)；Test(D)；Test(E)；Test(F)；Test(G)并且都测试通过。

非增量式集成测试强调一步到位式的集成测试，即只集成测试一次。

Test(A,B,C,D,E,F,G)

非增量式集成测试用例最少，测试过程简单，但测试的缺点也较为明显：当一次集成的模块较多时，非增量式测试容易出现混乱，因为测试时可能发现了许多故障，为每一个故障定位和纠正非常困难，并且在修正一个故障的同时，可能又引入了新的故障，新旧故障混杂，很难判定出错的具体原因和位置。

(2) 增量式集成测试

增量式集成测试是逐次将未曾集成测试的模块和已经集成测试的模块(或子系统)结合成程序包，再将这些模块集成为较大系统，在集成的过程中边连接边测试，以发现连接过程中产生的问题。按照不同的实施次序，增量式集成测试又可以分为以下三种不同的方法。

① 自顶向下增量式集成测试。自顶向下增量式测试表示逐步集成和逐步测试是按照程序结构图自上而下进行的，即模块集成的顺序是首先集成主控模块(主程序)，然后依照控制层次结构向下进行集成。从属于主控模块的其他模块按照深度优先方式(纵向)或者广度优先方式(横向)进行集成。

- 深度优先方式的集成测试。首先集成在结构中的一个主控路径下的所有模块，然后

集成其他路径下的所有模块。主控路径的选择是任意的,如图 2.5 所示。先集成 B 模块下的所有模块,再集成 C 模块,最后集成 D 模块下的所有模块。

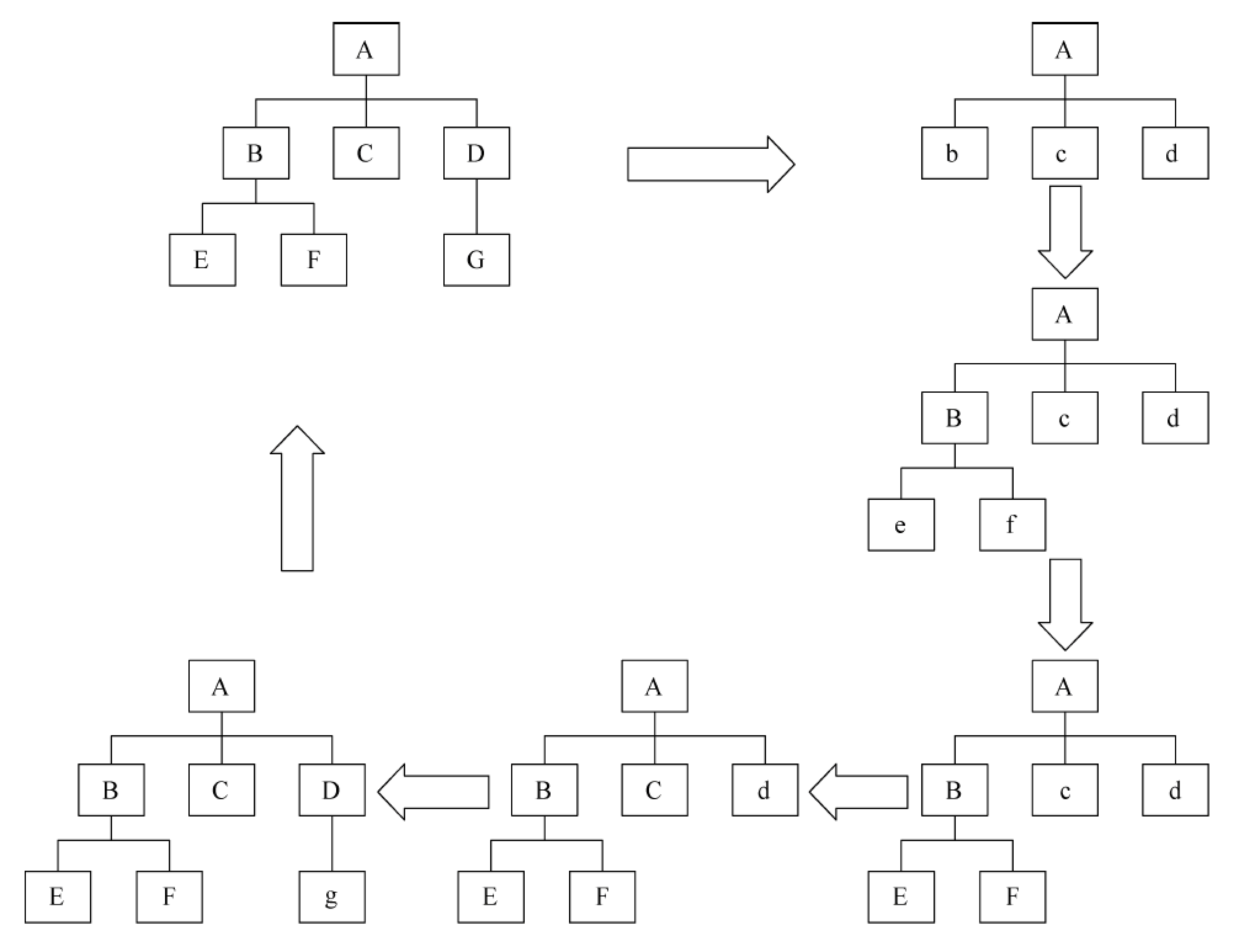


图 2.5　深度优先方式的集成测试图

• 广度优先方式的集成测试。首先沿着水平方向,把每一层中所有直接隶属于上一层的模块集成起来,直到集成到最底层,如图 2.6 所示。

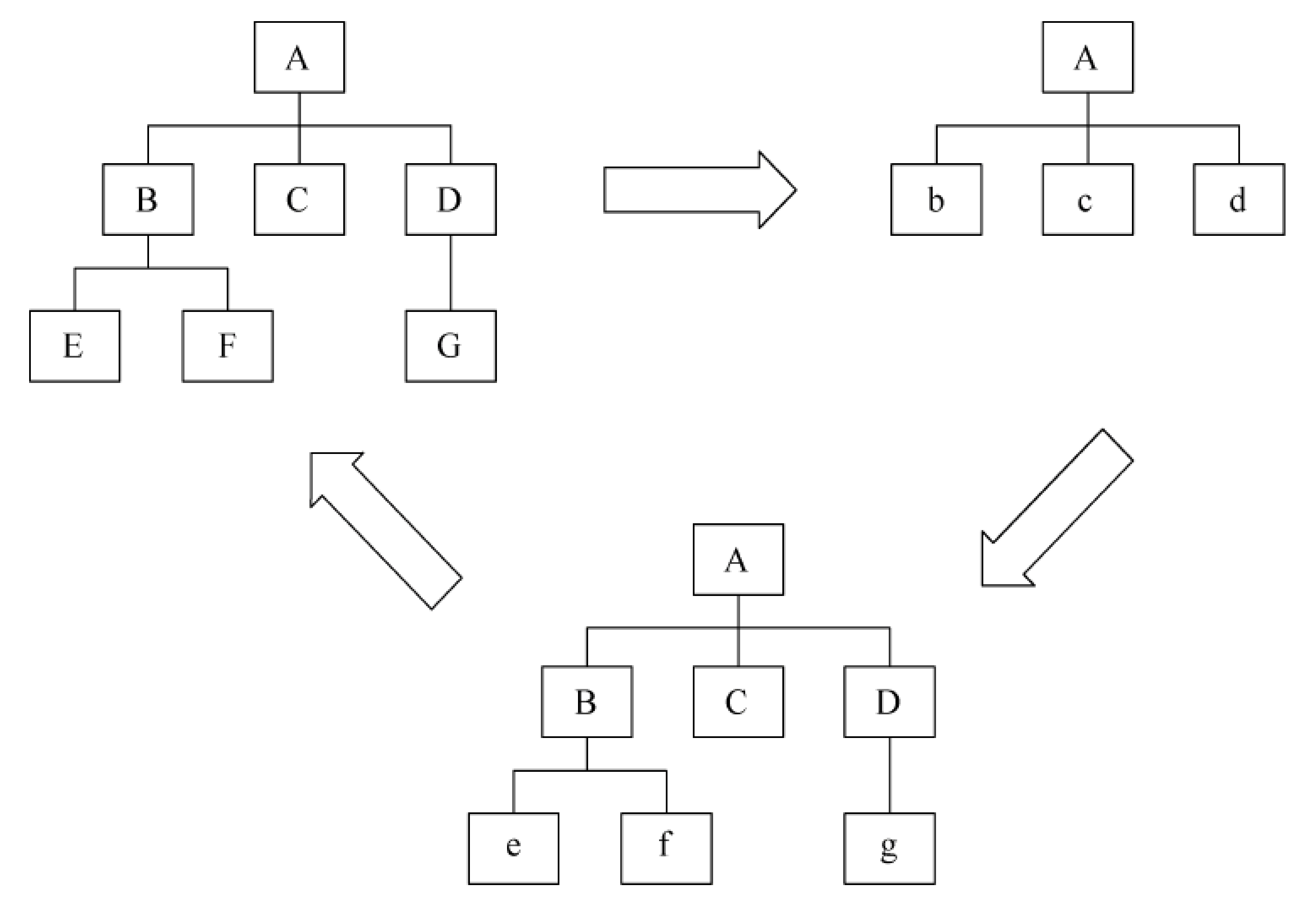


图 2.6　广度优先方式的集成测试图

自顶向下增量式集成测试的整个过程由以下几个步骤完成：主控模块作为测试驱动器；根据集成的方式(深度或广度)，下层的桩模块一次一次地被替换为真正的模块。在每个模块被集成时，都必须进行单元测试；重复第 2 步，直到整个系统被测试完成。

② 自底向上增量式集成测试。自底向上增量式集成测试表示逐步集成和逐步测试的工作是按结构图自下而上进行的，即从程序模块结构的最底层模块开始集成和测试。由于是从最底层开始集成，对于一个给定层次的模块，它的子模块(包括子模块的所有下属模块)已经集成并测试完成，所以不再需要使用桩模块进行辅助测试。

自顶向下与自底向上增量式测试的比较如表 2.3 所示。

表 2.3　自顶向下与自底向上增量式集成测试的比较

类别 比较	自顶向下增量式集成测试	自底向上增量式集成测试
优点	可以自然地做到逐步求精，一开始就能让测试者看到系统的整体框架	测试环境比较容易建立
缺点	需要提供桩模块，并且在输入、输出模块接入系统以前，在桩模块中表示测试数据有一定困难	直到最后一个模块被加进去之后才能看到整个系统的框架

(3) 混合增量式集成测试

混合增量式集成测试综合了自顶向下和自底向上两种集成方法，把系统划分成三层，中间一层为目标层，目标层上采用自顶向下集成，目标层下采用自底向上集成。

3) 系统测试

系统测试是将已经集成好的软件系统，作为整个基于计算机系统的一个元素，与硬件、支持软件、数据和人员等其他系统元素结合在一起，在实际运行环境下，对系统进行一系列的测试。系统测试的目的在于通过与系统的需求定义作比较，发现软件与系统的定义不一致的地方，以验证软件系统的功能和性能等方面是否满足系统的要求。

从软件测试的 V 模型来看，系统测试是产品提交给用户之前进行的最后阶段的测试，因此很多公司将其视为产品的最后一道防线。系统测试阶段主要使用黑盒方法设计测试用例。系统测试所用的数据必须尽可能地像真实数据一样精确和有代表性，也必须和真实数据的大小和复杂性相当。满足上述测试数据需求的一个最简便方法就是在系统测试中使用真实数据。系统测试对象为整个产品系统，它不仅包括产品系统的软件，还要包含系统软件所依赖的硬件、外设甚至包括接口。

系统测试依据为系统的需求规格说明书、各种规范。通信产品与一般的软件产品不同，其系统测试往往需要依据大量的既定规范。对于海外产品，系统测试依据还包括各个国家自定的规范。

系统测试根据测试的关注点不同，又可以分为功能测试、性能测试、可用性测试、压力测试、恢复测试、安全性测试、安装与卸载测试和兼容性测试等，分别介绍如下。

(1) 功能测试

功能测试是系统测试中最基本的测试，它不关心软件内部的实现逻辑，主要根据产品的

需求规格说明书和测试功能列表，验证产品的功能实现是否符合软件产品的需求规格。

（2）性能测试

性能测试是要检查系统是否满足在需求规格说明书中规定的性能。性能测试通常需要与压力测试结合起来，并要求同时进行硬件和软件检测。通常需要测试的性能信息包括：CPU 的使用情况、IO 使用情况、内存使用情况、每个模块执行时间百分比、系统响应时间、系统吞吐量等。

（3）可用性测试

可用性测试是指让一群有代表性的用户对产品进行典型操作，同时测试人员或开发人员在一旁观察、记录。可用性测试被用来改善软件的易用性，为用户提供一系列操作场景和任务让他们去完成，从而发现使用过程中出现了的问题、用户喜欢或不喜欢哪些功能和操作方式、原因是什么等等。针对问题所在，提出改进建议。测试人员和程序员通常不宜做可用性测试。可用性测试中最重要的就是 UI（User Interface）测试，即用户跟计算机交互。

（4）压力测试

压力测试用来测试系统在其资源超负荷情况下的运行状况。如成千上万的用户同时登录某系统，短时间内引入超负荷的数据容量，同时引入大量的用户操作。压力测试中通常采用自动化测试工具来完成。如后面将学到的 LoadRunner 就是业界常用的压力测试自动化工具。在压力测试时，尽量让压力依次增大，直到系统中断，并且需要重复进行测试，才能确定系统在什么样的负荷下正常运行。

（5）恢复测试

恢复测试是指验证系统从软件或硬件失败中恢复系统的能力。可采用各种人工干预的手段，模拟硬件故障，故意造成软件出错等，并由此检查：错误探测功能，即系统能否发现硬件失效与故障；能否切换或启动备用的硬件设备；在故障发生时能否保护正在运行的作业和系统状态；在系统恢复后能否从最后记录下来的无错误状态开始继续执行作业；掉电测试：其目的是测试软件系统在发生电源中断时能否保护当时的状态且不毁坏数据，然后在电源恢复时从保留的断点处重新进行操作。

（6）安全性测试

安全性测试用来验证集成在系统内的保护机制是否能够在实际中保护系统不受到非法用户的入侵。包括：系统层安全测试，如系统中是否存在不必要的用户账号、文件和目录权限、日志文件和口令策略等；网络安全层测试，如在数据通信和数据交互过程中，对数据进行截取分析等；应用安全层测试，如身份验证、权限管理等。

（7）安装和卸载测试

安装测试的目的不是找软件错误，而是找安装错误。安装程序错误可能源于以下几个方面：环境的检测（如，有多少可用磁盘空间）、系统和环境配置、软件和硬件的兼容性等。

卸载测试的目的就是验证成功卸载系统的能力。在卸载程序过程通常会有以下活动：删除目录；删除应用程序的 EXE 文件和专用 DLL 文件；检查特定文件是否被其他已安装的应用程序使用；如果没有其他应用程序使用，删除共享文件；删除注册表信息；恢复原有注册表项；通过添加或删除程序执行卸载等。

(8) 兼容性测试

兼容性测试用来测试系统对其他应用或者系统的兼容性。兼容性测试考虑以下问题：硬件兼容性、浏览器兼容性、数据库兼容性、操作系统兼容性等。

4) 验收测试

验收测试是部署软件之前的最后一个测试操作。它用来确保软件准备就绪，向用户表明系统能够像预定要求的方式工作，也就是验证软件的有效性。有效性的保证要根据软件需求规格说明书和验收要求编制《验收测试计划》，制定需测试的测试项，制定测试策略及《项目验收准则》，并经过客户参与计划评审。

实施验收测试的常用策略有三种，它们分别是：正式验收测试、非正式验收测试和 β 测试。选择的策略通常建立在合同需求、组织和公司标准以及应用领域的基础上。

(1) 正式验收测试

正式验收测试是一项管理严格的过程，它通常是系统测试的延续。计划和设计这些测试的周密和详细程度不亚于系统测试。选择的测试用例应该是系统测试中所执行测试用例的子集。可以采用两种方式：开发组织(或其独立的测试小组)与最终用户一起执行验收测试；验收测试完全由最终用户组织执行，或者由最终用户选择人员组成一个客观公正的小组来执行。

(2) 非正式验收测试

在非正式验收测试中，执行测试过程的限定不像正式验收测试中那样严格。在此测试中，确定并记录要研究的功能和业务任务，但没有可以遵循的特定测试用例。测试内容由各测试员决定。这种验收测试方法不像正式验收测试那样组织有序，有很大的主观随意性。大多数情况下，非正式验收测试是由最终用户组织执行的。

(3) β 测试

β 测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用 β 版本，并要求用户报告异常情况、提出批评意见，然后软件开发公司再对 β 版本进行改错和完善。

2. 软件测试过程 W 模型

V 模型也存在着严重的缺陷，该模型主要是针对程序进行测试寻找错误，而需求分析阶段隐藏的问题一直到后期的验收测试才被发现，这种问题往往是致命的，最终导致软件产品的失败。容易使人误解测试是软件开发的最后一个阶段，而不是伴随在软件产品的整个生命周期中。

W 模型认为软件测试伴随着整个软件开发周期，而且测试的对象不仅是程序，需求、功能和设计同样需要测试。这样，只要相应的开发活动完成，就可以开始执行测试，即测试与开发是同步的，有利于尽早地发现问题。

但是 W 模型和 V 模型都是把软件的开发视为需求、设计、编码等一系列串行的活动。需要有严格的指令表示上一阶段完全结束，才可正式开始下一个阶段。这样就无法支持迭代、自发性以及变更的调整。W 模型图如图 2.7 所示。

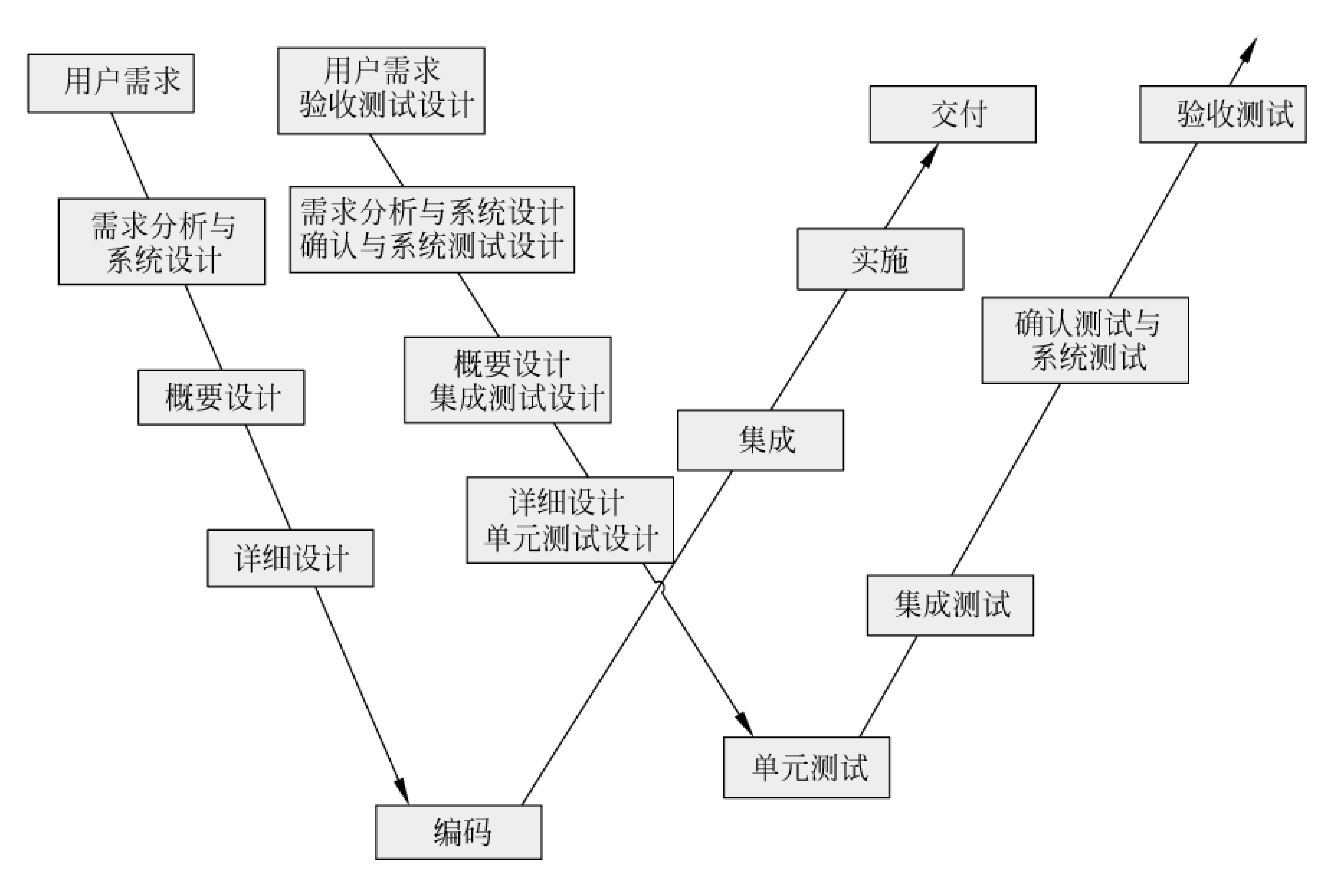


图 2.7　软件测试 W 模型图

3. 软件测试过程 H模型

H 模型相对 W 模型和 V 模型来说没有严格的活动结束标志区分,更加的灵活多变。该模型强调软件开发活动之间存在互相牵制的关系,它们是可以交叉进行的。所以,相应的测试之间也不存在严格的次序关系。同时,各层次之间的测试也存在反复触发、迭代和增量关系。H 模型将测试活动完全独立出来,形成一个完全独立的流程,将测试准备活动和测试执行活动清晰地体现出来。H 模型图如图 2.8 所示。

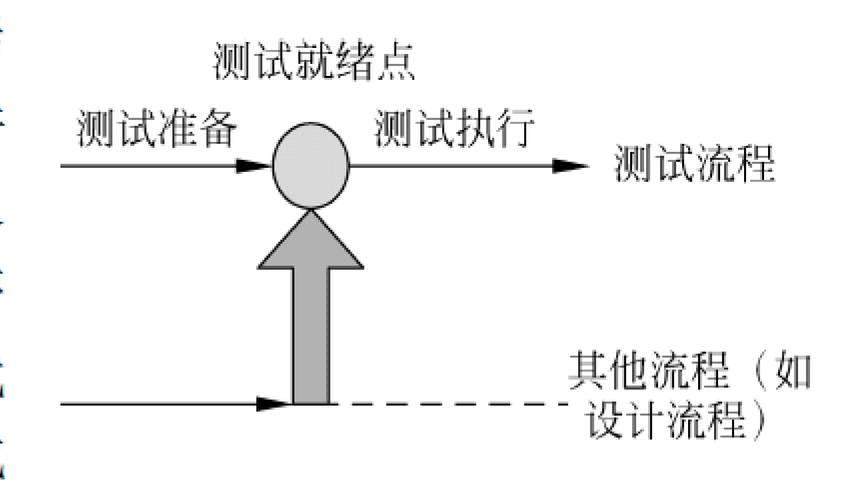


图 2.8　软件测试 H 模型图

2.2.2　能力目标

掌握软件测试的三种模型：V 模型、W 模型和 H 模型,以及三种模型的区别,重点掌握 V 模型；在 V 模型中测试阶段分为：单元测试、集成测试、系统测试和验收测试四个阶段,这四个阶段分别和软件开发阶段如何对应；掌握单元测试的基本概念,了解桩模块和驱动模块；掌握集成测试的基本概念和主要方法；了解系统测试和验收测试的基本概念。

2.2.3　任务驱动

1. 在单元测试中,测试单元是如何划分的？单元测试由几部分组成？

2. 如果开发时间紧迫,是否可以跳过单元测试而直接进行集成测试？如果不可以,请说明原因。

3. 集成测试的策略有几种？

4. 系统测试主要包括哪些内容？

5. 验收测试是如何分类的?

2.2.4 实践环节

1. 创建一个待测试程序计算器类,类名为 Calculator。
2. 创建方法 public int add(),实现两个整数的加法运算。
3. 创建方法 public int substract(),实现两个整数的减法运算。
4. 创建测试类 Test 对 Calculator 进行单元测试。
5. 指出这两个类中的驱动模块、桩模块和被测模块。

2.3 软件测试工作流程

2.3.1 核心知识

在了解了软件测试一些基础知识后,测试人员该如何展开测试工作呢?软件测试工作的流程大体包括 5 个部分,分别是测试需求分析、制订测试计划、设计测试用例、测试用例的执行、进行测试评估。具体工作流程如图 2.9 所示。

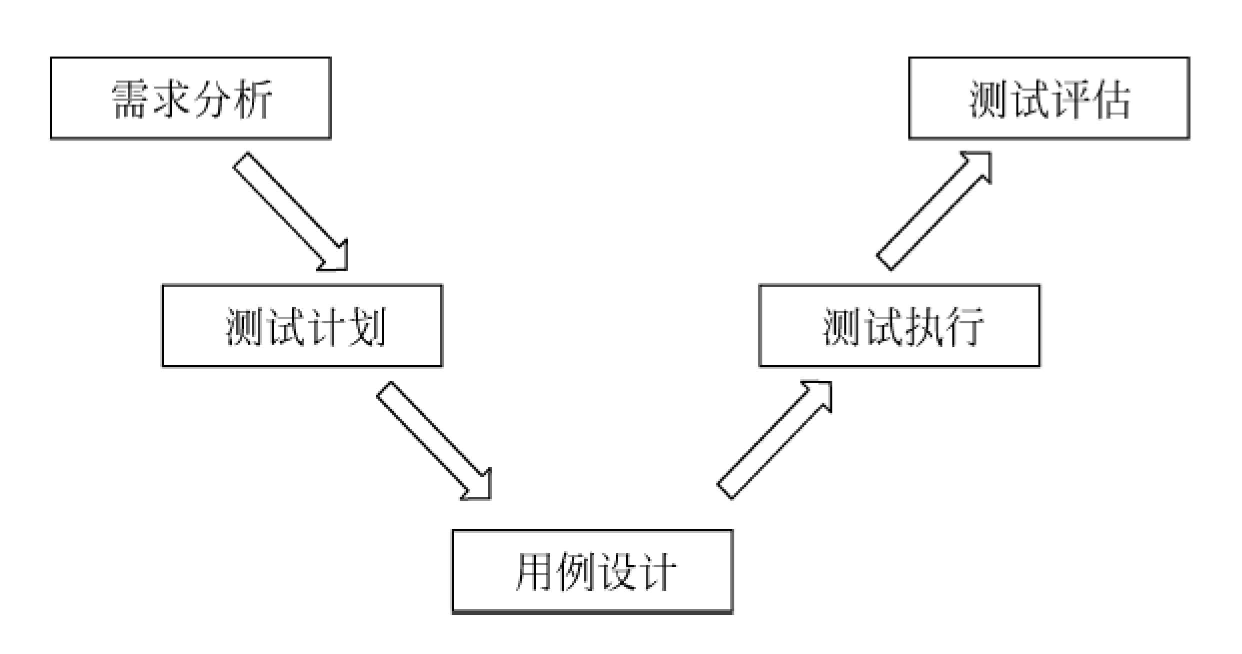


图 2.9 软件测试工作流程图

1. 测试需求分析

图 2.9 中的需求分析指的是测试需求分析。包括软件功能需求分析、测试环境需求分析、测试资源需求分析等。测试需求分析的主要目的是对需求设计的理解和用户需求的理解,把不直观的需求转变为直观的需求,把不明确的需求转变为明确的需求,把不能度量的需求转变为可度量的需求,从而进行测试设计。而通常所说的需求分析指的是软件功能需求分析,它是指理解用户需求,就软件功能与客户达成一致,估计软件风险和评估项目代价,最终形成开发计划的一个复杂过程。软件功能需求分析的主要目的是对需求设计的理解,进行需求设计。

测试需求分析包含以下 4 个阶段。

(1) 明确需求的范围。确定需求中功能点的个数、粒度、优先级以及流向等。

(2) 明确每一个功能的业务处理过程。确定每个功能的输入、处理和输出。

(3) 不同的功能点的组合。各功能点间的耦合性。

(4) 挖掘显式需求背后的隐式需求。挖掘非功能性需求。

测试需求分析阶段的产物包括:

(1) 测试整体架构(测试计划);
(2) 功能点检证表;
(3) 功能处理流程图。

2. 测试计划

测试计划(Testing Plan),描述了要进行的测试活动的范围、方法、资源和进度的文档。它确定测试项、被测试项的特性、测试任务、有谁来执行测试任务、测试中可能具有的各种风险。测试计划可以有效预防计划阶段的风险,保障计划的顺利实施。

编写测试计划需要解决以下 6 个问题,即测试计划的 6 个要素。

(1) WHY:为什么要进行这些测试;
(2) WHAT:测试哪些方面,不同阶段的工作内容;
(3) WHEN:测试不同阶段的起止时间;
(4) WHERE:相应文档,缺陷的存放位置,测试环境等;
(5) WHO:项目有关人员组成,安排哪些测试人员进行测试;
(6) HOW:如何去做,使用哪些测试工具以及测试方法进行测试。

测试计划的主体内容如表 2.4 所示。

表 2.4 测试计划主体内容

测试背景	测试依据	测试资源	测试策略	测试日程	测试标准
软件项目介绍;项目涉及人员,如软、硬件项目负责人等	软件需求文档;软件规格书;软件设计文档;其他附件,如参考产品等	测试设备需求;测试人员需求;测试环境需求	采取何种测试方法;搭建哪些测试环境;采取哪些测试工具以及测试管理工具;对测试人员进行培训等	测试需求分析与评审;测试用例编写;测试实施,测试分成哪些测试阶段,每个阶段的工作重点以及投入资源等;测试评估报告	测试阶段标准(阶段目标与质量);测试结束标准

测试计划模板案例

软件测试计划

一、概述

1.1 编写目的

此计划编写的目的是为使移动交易平台 V1.0 版能够达到与系统说明书所描述的功能一致,并且检验系统是否运行稳定。

1.2 参考资料

(1)《移动交易平台系统需求分析说明书》
(2)《移动交易平台系统系统分析说明书》

1.3 背景

近年来,伴随着移动互联业务的不断拓展,移动服务提供商(移动相关的其他增值服务提供商,Service Provider)提供的服务也越来越丰富,其中包括短信购物、信息定制(气象预报等)、手机网络游戏、手机理财等各种服务。SP 业务一般都是由第三方运营的,为了简化项目模型和项目规模,特此推出移动交易平台 V1.0 版。

二、约定

2.1　测试的目的和任务

目的：完成整个系统的测试及验证软件的基本可用性，功能的完整性，数据的准确性等。

任务如下：

(1) 与《移动交易平台系统需求分析说明书》、《移动交易平台系统系统分析说明书》比较，检查此软件所完成的功能，是否与上面两个说明书相符合；

(2) 数据业务是否能够正确完成；

(3) 整个系统是否能够稳定地运行；

(4) 帮助等其他安装说明文件是否表达准确。

2.2　人员和设备

(1) 人员：

测试人员：**

编程人员：***

(2) 设备：

Web 服务器(WebLogic8.0)　IP 地址：192.168.62.39

数据库服务器(Oracle10g)

2.3　送测要求

移动交易平台系统开发人员提交的测试，按以下要求进行。

步骤	动作	负责人	相关文档或记录	要　　求
1	打包、编译	***	无	确认可测试
2	接收测试	***	用户需求说明书 系统设计说明书	确认开始测试
3	开始测试	***	Bug 单、小结	测试小结（个人编写个人的内容）

2.4　编号规则

(1) 测试用例中的编号，模块名＋界面名＋编号

(2) 测试用例文件命名规则，模块名＋测试用例

2.5　测试的安排和进度

进度安排如下。

测试阶段	测 试 任 务	工作量估计	人员分配	起止时间
第一阶段 功能测试	1. 手机用户登录和退出 2. 消息接收与发送 3. 话费余额查询 4. 话费充值 5. 定制天气预报 6. 通用模块 …… 是否能正确实现其功能，是否有操作错误	10 日	***	120720—120730

续表

测试阶段	测 试 任 务	工作量估计	人员分配	起止时间
第二阶段 系统测试	1. 完成所有模块的组合测试 2. 确定所有业务流向和数据的正确性	5 日	***	120801—120805
第三阶段 性能测试	在多用户访问,交替进行压力和性能测试	2 日	***	120806—120807
第四阶段 安装手册帮助文件测试以及安装卸载测试	1. 将安装手册和用户帮助手册与软件操作比较是否有不符 2. 对安装文件进行测试	2 日	***	120808—120809
第五阶段 兼容测试	软件在各个软件平台上的运行情况	1 日	***	120810

三、测试种类及测试标准

3.1　功能测试阶段

(1) 功能测试：测试各个模块以及窗口所完成的功能是否准确,数据是否正确,操作是否简洁方便。

(2) 功能键及界面测试：功能键是否描述准确、操作方便,界面是否设计简洁、符合用户需求说明。

(3) 数据项测试：

① 输入正确数据是否能按照预期的答案回显；

② 是否能识别错误的输入数据,并给予正确的信息提示。

3.2　系统测试阶段

(1) 业务流程测试：按照系统分析说明书的业务流程,检查本系统所完成的业务流程是否正确。

(2) 数据流测试：本系统所涉及的相关数据,是否按照正确的业务流程流动,每个阶段所反映的数据结果是否正确。

3.3　性能测试阶段

模拟客户进行多用户测试,压力测试有一条 8∶2 原则,即 80%的业务量在 20%的时间内输入。

3.4　安装手册帮助文件测试以及安装卸载测试阶段

(1) 帮助文件的测试：

① 帮助文档是否精确描述了如何使用各种功能；

② 举例是否精确；

③ 术语、菜单描述和系统响应是否与实际程序一致；

④ 是否能够很方便地在文档中定位指南；

⑤ 是否能够很方便地使用文档排除错误。

(2) 安装卸载测试阶段：

① 自动安装还是手工配置安装,测试各种不同的安装组合,并验证各种不同组合的正确性,最终目标是所有组合都能安装成功；

② 安装退出之后，确认应用程序可以正确启动、运行；

③ 卸载测试和安装测试同样重要，如果系统提供自动卸载工具，那么卸载之后需检验系统是否把所有的文件全部删除，注册表中有关的注册信息是否也被删除；

④ 安装时间是否合理；

⑤ 对于客户端服务器(C/S)模式的应用系统，可以先安装客户端，然后安装服务器端，测试是否会出现问题。

3.5 兼容测试阶段

验证本软件在几种常用的操作系统下的运行情况，并且检查本软件与其他软件并行时是否运行正确。

3.6 重点测试部分

SP 服务定制部分。

3.7 测试设计

详见测试用例设计说明书。

四、测试提交物

本次测试完成后的提交物：

(1) 测试计划；

(2) 测试用例；

(3) 测试 Bug 单。

3. 测试用例设计

测试用例是为特定的目的而设计的一组测试输入、执行条件和预期的结果的集合。测试用例是执行的最小实体。简单地说，测试用例就是设计一个场景，使软件程序在这种场景下，必须能够正常运行并且达到程序所设计的执行结果。

1) 测试用例的重要性

确定测试用例之所以很重要，原因在于测试用例构成了设计和制定测试过程的基础；判断测试是否完全的一个主要评测方法是基于需求的覆盖，而这又是以确定、实施和执行的测试用例的数量为依据；测试工作量与测试用例的数量成比例。根据全面且细化的测试用例，可以更准确地估计测试周期各连续阶段的时间安排是否合理。测试设计和开发的类型以及所需的资源主要都受控于测试用例。

2) 设计测试用例的基本准则

- 测试用例的代表性。能够代表并覆盖各种合理的和不合理的、合法的和非法的、边界的和越界的以及极限的输入数据、操作和环境设置等。
- 测试结果的可判定性。即测试结果的正确性是可判定的，每一个测试用例都应有相应的期望结果。
- 测试结果的可再现性。即对同样的测试用例，系统的执行结果应当是相同的。

3) 测试用例质量

测试用例质量考核的标准主要有 4 个：有效性、可效仿性、经济性和修改性。

- 有效性主要指的是测试用例是否可以发现软件的缺陷、至少可能发现软件的缺陷；
- 可效仿性主要指测试用例可以测试多项内容，因而可以减少测试事例的数量；
- 经济性主要指测试用例在测试执行、分析和调试方面是否经济；

- 修改性主要指测试用例在日后的维护方面是否易于修改。

4）测试用例组成元素

- 用例 ID；（唯一性）
- 用例名称；（明示测试部分）
- 测试目的；
- 测试级别；
- 参考信息；（测试用例设计依据）
- 测试环境；
- 前提条件；
- 测试步骤；
- 预期结果；
- 设计人员。（更新维护）

5）测试用例的设计方法

（1）黑盒测试

① 等价类划分法，是把所有可能的输入数据，即程序的输入域划分成若干部分（子集），然后从每一个子集中选取少数具有代表性的数据作为测试用例。

② 边界值分析法，就是对输入或输出的边界值进行测试的一种黑盒测试方法。

③ 决策表分析法，是分析和表达多逻辑条件下执行不同操作情况的工具。

④ 因果图法，采用一种适合于描述多种条件的组合、相应产生多个动作的形式来进行测试用例设计。

⑤ 错误推算法，基于经验和直觉推测程序中所有可能存在的各种错误，从而有针对性地设计测试用例的方法。

（2）白盒测试

① 逻辑覆盖，是以程序内部的逻辑结构为基础的设计测试用例技术。又分为：语句覆盖、判定覆盖、条件覆盖、判定—条件覆盖、条件组合覆盖、路径覆盖。

② 独立路径测试，该方法把覆盖的路径数压缩到一定限度内，程序中的循环体最多只执行一次（以程序控制流图为基础）。

③ 循环结构分析，分为 4 种不同类型：简单循环、连锁循环、嵌套循环和非结构循环。

4. 测试执行

测试执行一般会经历以下 3 个阶段，如图 2.10 所示。

（1）初测期：测试主要功能和关键执行路径，排除主要障碍。

（2）细测期：依据测试计划和测试用例，逐一测试功能、性能、用户界面、兼容性、可用性等。

（3）回归测试期：复查已知错误的纠正情况，当确认未引发任何新的错误时，可以终结回归测试。

5. 测试评估

测试评估阶段的主要产物是测试报告。测试报告的目的是总结当前的软件测试工作，对被测试软件的版本质量作出评估，给产品能否发布提供一个参考值。

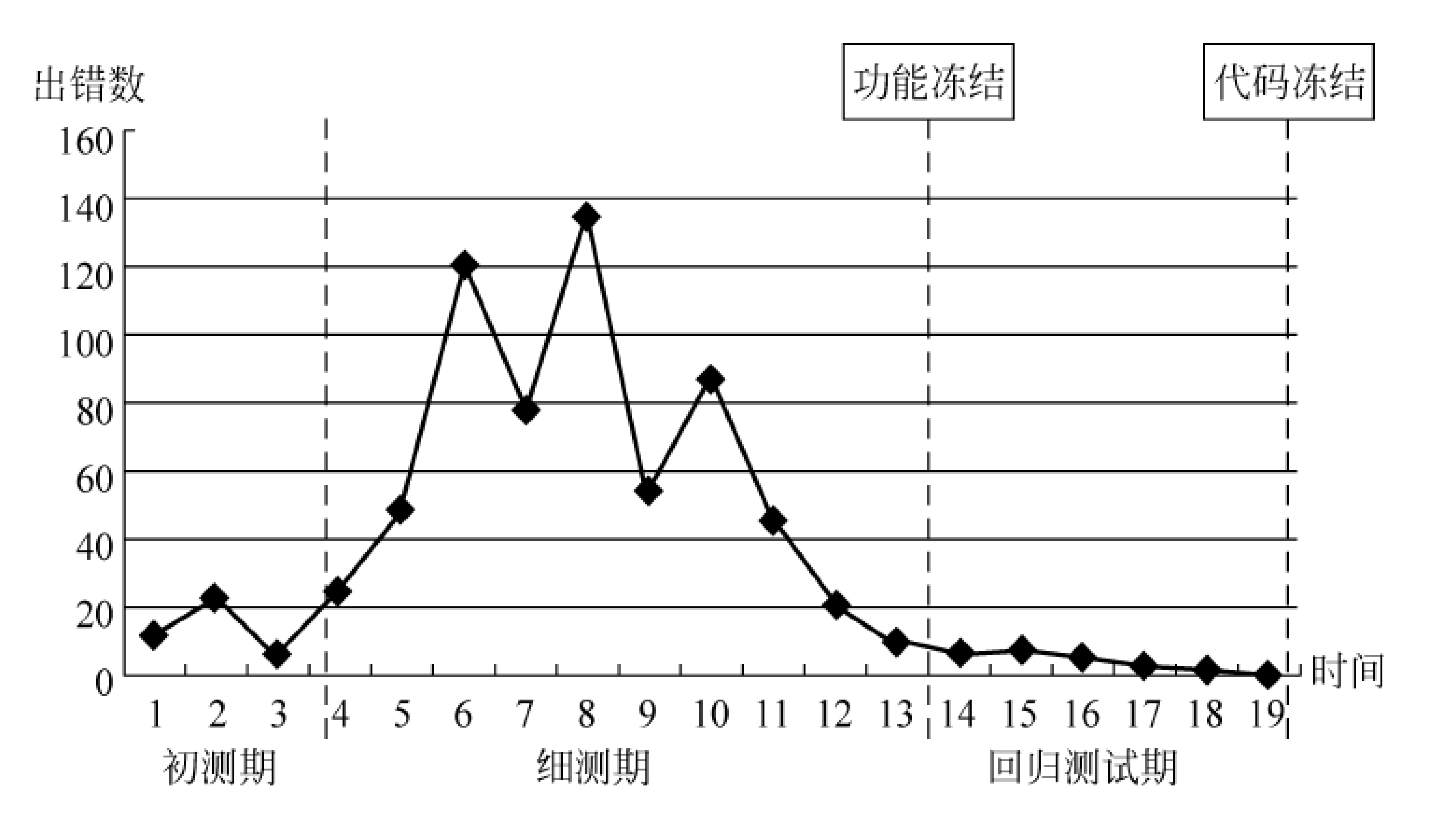


图 2.10 测试执行期三个阶段图

软件测试报告是软件测试过程中最重要的文档，它的内容包括：记录问题发生的环境，如各种资源的配置情况；记录问题的再现步骤；记录问题性质的说明；记录问题的处理进程。

2.3.2 能力目标

了解软件测试流程的主要步骤；掌握测试需求分析的 4 个阶段，了解需求分析阶段的成果物；掌握软件测试计划的主要内容，能够套用模板编写测试计划；掌握测试用例设计的基本原则和主要方法；掌握测试计划的主体内容，能够制订测试计划；了解测试执行的三个阶段；了解测试评估的作用。

2.3.3 任务驱动

1. 软件测试需求分析包括哪些内容？
2. 测试计划需要解决哪些问题？
3. 没有测试计划，是否可以进行软件测试工作？
4. 测试评估阶段的产物是什么？
5. 什么时候可以停止测试？有何标准？

2.3.4 实践环节

1. 通过查询资料，说明软件测试需求分析与软件设计需求分析的区别，各自的侧重点是什么？
2. 针对微信软件，进行该软件的功能需求分析。
3. 按照本节中的测试计划模板，针对学生成绩管理系统设计测试计划。

2.4 面向对象的软件测试

2.4.1 核心知识

面向对象技术是一种全新的软件开发技术，正逐渐代替被广泛使用的面向过程开发方法，被看成是解决软件危机的新兴技术。面向对象技术产生更好的系统结构，更规范的编程

风格，极大地优化了数据使用的安全性，提高了程序代码的重用，一些人就此认为面向对象技术开发出的程序无须进行测试。应该看到，尽管面向对象技术的基本思想保证了软件应该具有更高的质量，但实际情况却并非如此，因为无论采用什么样的编程技术，编程人员的错误都是不可避免的，而且由于面向对象技术开发的软件代码重用率高，更需要严格测试，避免错误的繁衍。因此，软件测试并没有因为面向对象编程的兴起而丧失掉它的重要性。

1. 面向对象技术核心概念

现在普遍比较认可的观点认为，面向对象技术主要包括 6 个核心概念：对象、消息、接口、类、继承、多态。

（1）对象

对象是一个可操作的实体，它既包含了特定的数据，也包含了操作这些数据的代码。在面向对象的程序设计中，对象是一个基本的可计算实体，对象被创建、修改、访问或由于协作的结果而被删除。在一个良好的面向对象的设计理念中，程序中的对象是一些问题及其解决方法的特定实体的描述。在程序中，对象之间存在着一定的联系；而在问题领域中，就可能反映出它们的相对关系。

对象是软件开发期间测试的直接目标。在程序运行时，对象的行为是否符合它的说明规定，该对象与和它相关的对象是否协同工作，这两方面是面向对象软件测试所关注的焦点。对象可以用它的生命周期来描述。当一个对象被创建时它的生命周期就开始了，这个过程贯穿于对象的一系列状态，当一个对象被删除时它的生命周期也就结束了。

从测试视角的角度，关于对象可以得到如下观点。

① 对象的封装。这使得已定义的对象容易识别，在系统中容易传递，也容易操纵。

② 对象隐藏信息。这一点使得对象信息的改变有时很难观察到，因此也使得测试结果检查的难度加大。

③ 对象的状态。在对象的生命周期中，对象都有一个状态。对象的状态是多变的，因此也可能是不正常行为的根源。

④ 对象的生命周期。对象都具有生命周期。在对象生命周期的不同阶段，为了确定对象的状态是否符合它的生命周期，对象可能会被从各个方面进行检测。过早地创建一个对象或过早地删除一个对象，都是造成错误的常见原因。

（2）消息

消息是对象的操作将要执行的一种请求。除了需要一个操作的名字，消息还可包含一些值（实参），它们常常在操作被执行时使用。消息的接收者也可以将某个值返回给消息的发送者。

面向对象的程序是通过一系列对象协同工作来解决问题的，这一协作是通过对象之间互相传送消息来完成的。我们把发送消息的对象称为发送者，把接收消息的对象称为接收者。有一些消息会以不同的形式返回结果，例如接收者以返回值或者异常的形式把结果传送给发送者。

面向对象程序执行的典型过程首先是实例化对象，然后将一条消息传送给其中的某个对象，消息的接收者把它自己产生的消息发送给其他对象（甚至是发送给自己）来执行计算。比如在一些由事件驱动的环境下，环境会不断地发送消息并且等待诸如鼠标点击和按键等外部事件的响应。

从测试视角的角度，关于消息有如下结论。

① 消息发送者。发送者可以决定何时发送消息，并且可能会做出错误的决定。

② 消息接收者。接收者可能接收到非预期的特定消息。当它接收到一条非预期的消息时，接收者可能会对此做出不正确的反应。

③ 消息可以含有实际参数。在处理一条消息时，参数能被接收者使用或更改。如果传递的参数是对象，那么在消息被处理前和处理后，对象必须处于正确的状态，而且它们必须实现接收者所期望的接口。

(3) 接口

接口是行为声明的集合。行为被集中在一起，并通过单个的概念定义一些相关的动作。接口是由一些规范构成的，这一规范定义了类的一套完整的公共行为。通过定义一个抽象的基类，其内部仅包含公有的抽象方法，可以达到定义接口的目的。

从测试视角的角度，关于接口有如下结论。

① 接口封装了操作的说明。这些说明逐步形成了对实现类的规范。如果类的行为和接口包含的行为不相符时，该类就不能实例化，即不能创建对象。

② 接口不是孤立的，它与其他的接口和类有一定的关系。一个接口可指定一个行为的参数类型，使得实现该接口的类可被当作一个参数进行传递。

(4) 类

类是一组具有相同数据结构和相同操作的对象的集合。类的定义包括一组数据属性和在数据上的一组合法操作。在面向对象的程序中，任何被描述的概念最初都必须被声明为类，然后创建由该类定义的对象。创建对象的过程被称作实例化，而创建的结果被称为实例。在一个类中，每个对象都是类的实例，它们都可使用类中提供的方法。一个对象的状态则包含在它的实例变量中。

从测试视角的角度来研究类，可以得出下列设计和实现中潜在错误的原因。

① 类的说明包含用来构造实例的一些操作，这些操作可能会导致不正确的初始化新实例的属性。

② 类在定义自己的行为和属性时，依赖于跟它共同协作的其他类。如果在类的定义中使用了包含有不正确实现的其他类，这就会使类发生错误。

③ 类本身的说明，并不一定是正确的。

④ 类的实现可能不支持所有要求的操作，或者执行一些错误的操作。

⑤ 类需要指定每个操作被执行之前应满足的条件。在发送者发送一条消息之前，它可能没有提供检验这些条件的方法。

(5) 继承

继承是类之间的一种联系，它允许新类可以在一个已有类的基础上进行定义。一个类对另一个类的依赖，使得已有类的说明和实现可以被复用。这种方法有一个重要的优势，那就是已有类不会被改变，并且对其他任何继承它的新类来说都是一样的。这里所说的新类主要是指子类或派生类，被新类继承的已有类就叫基类。一个基类，以及从这个基类直接或间接继承而得到的派生类，它们共同构成继承层次关系。在继承层次关系中，把这个基类叫做“根”，派生类则是从根直接或间接继承过来的。除了根，每个类都有一到多个直接或间接继承的祖先，每个类也都有若干从它继承过去的派生类。

从测试视角的角度来看，继承包含以下内容。

① 继承提供了一种机制，通过这种机制，潜在的错误能够从一个类传递到它的派生类中。测试类的时候要尽早消除这种错误，以免这些错误一并传递到其他类中。

② 继承机制能使子类重复使用相同的测试方法。因为子类是从它的父类继承过来的，所以子类也就继承了父类的说明和实现。因此就可以用测试父类的方法对子类进行测试。

(6) 多态

多态提供了将对象看作是一种或多种类型的能力。编程语言的类型机制可用来支持许多不同的类型适应策略。此外类型的完全匹配可能是最安全的，但多态却支持灵活的设计，同时又易于维护。多态有几种不同的形式，如参数多态、包含多态、过载多态。参数多态是能够根据一个或多个参数来定义一种类型的能力。包含多态和过载多态在面向对象语言中通常体现在子类与父类的继承关系上。

包含多态是同一个类具有不同表现形式的一种现象。面向对象的编程语言对包含多态的支持(也有人把这种支持叫做动态绑定)，使得参数具有对象替换的能力。为了响应操作请求，当对象的定义与后续对象的定义相符时，对象就可以被相互替换。换句话说，面向对象程序中的发送者能够在用对象作为参数时根据接口进行实现，而不是实现一个完整的类。包含多态提供了一种强有力的能力，可以任意设计接口并编写代码，而不用考虑其他类的对象是否发送了消息来请求操作。包含多态使得设计和编码比以前更加抽象。实际上，定义一个没有实例的抽象类是非常有用的，而它的子类则有实例存在。抽象类主要是为了定义一个由所有它的派生类支持的接口。

从测试视角的角度来看，包含多态具有以下功能。

① 包含多态允许系统通过增加类来进行扩展，而不用修改已经存在的类。

② 包含多态允许任何操作都可以包括一个或多个类型不确定的参数，如使用接口作为参数。

③ 包含多态允许操作指定动态引用返回的响应。

2. 面向对象测试模型

面向对象的开发模型突破了传统的瀑布模型，将开发分为面向对象分析(OOA)、面向对象设计(OOD)和面向对象编程(OOP)三个阶段。分析阶段产生整个问题空间的抽象描述，在此基础上，进一步归纳出适用于面向对象编程语言的类和类结构，最后形成代码。由于面向对象的特点，采用这种开发模型能有效地将分析设计的文本或图表代码化，不断适应用户需求的变动。与这种开发模型对应的面向对象测试模型包括：面向对象分析的测试(OOA Test)、面向对象设计的测试(OOD Test)、面向对象编程的测试(OOP Test)、面向对象单元测试(OO Unit Test)、面向对象集成测试(OO Integrate Test)、面向对象系统测试(OO System Test)。面向对象测试模型如图 2.11 所示。

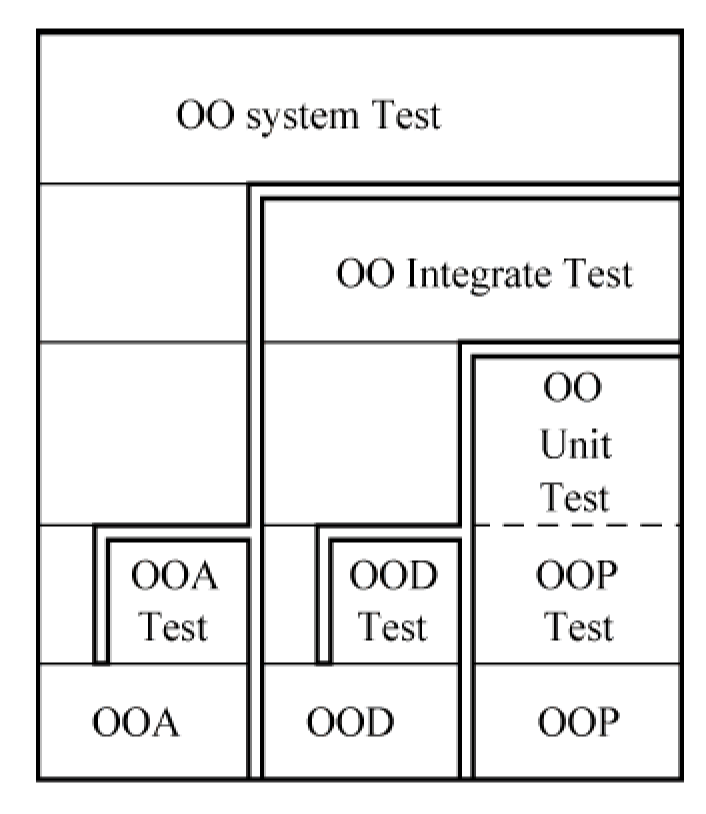


图 2.11　面向对象测试模型

OOA Test 和 OOD Test 是对分析结果和设计结果的测试，主要是对分析设计产生的文本进行，是软件开发前期的关键性测试。OOP Test 主要针对编程风格和程序代

码实现进行测试，其主要的测试内容在面向对象单元测试和面向对象集成测试中体现。

传统测试模型与面向对象的测试模型最主要的区别在于：面向对象的测试更关注对象，而不是完成输入或输出的单一功能，这样，测试可以在分析与设计阶段就先行介入，使得测试更好地配合软件生产过程并为之服务。与传统测试模式相比，面向对象测试的优点在于：更早地定义出测试用例；早期介入可以降低成本；尽早地编写系统测试用例以便于开发人员与测试人员对系统需求的理解保持一致；面向对象的测试模式更注重于软件的实质。

（1）面向对象单元测试

面向对象单元测试是对程序内部具体单一的功能模块的测试，如果程序是用 Java 语言实现，主要就是对类成员方法的测试。面向对象单元测试是进行面向对象集成测试的基础。

（2）面向对象集成测试

面向对象集成测试也叫类簇测试。类簇是指一组相互有影响，联系比较紧密的类。它是一个相对独立的实体，在整体上是可执行和可测试的，并且实现了一个内聚的责任集合，但不提供被测试程序的全部功能，相当于一个子系统。类簇测试主要根据系统中相关类的层次关系，检查类之间的相互作用的正确性，即检查各相关类之间消息连接的合法性、子类的继承性与父类的一致性、动态绑定执行的正确性、类簇协同完成系统功能的正确性等。

面向对象集成测试又可以分为两种。

① 基于类间协作关系的横向测试。由系统的一个输入事件作为源头，对其触发的一组类进行测试，执行相应的操作和消息处理路径，最后终止于某一输出事件。应用回归测试对已测试过的类簇再重新执行一次，以保证加入新类时不会产生意外的结果。

② 基于类间继承关系的纵向测试。首先通过测试不使用或很少使用其他类服务的类，即独立类(是系统中已经测试正确的某类)来开始构造系统。在独立类测试完成后，下一层继承独立类的类(称为依赖类)被测试，这个依赖类层次的测试序列一直循环执行到构造完整个系统。

（3）面向对象系统测试

面向对象系统测试是基于面向对象集成测试最后阶段的测试，系统测试是对所有程序和外部成员构成的整个系统进行整体测试，检验软件和其他系统成员配合工作是否正确。另外，还包括了确认测试内容，以验证软件系统的正确性和性能指标等是否满足需求规格说明书所制定的要求。

3. 面向对象软件测试的整体流程

（1）面向对象的单元测试。编码人员进行单元测试工作，以规范的格式提交可运行的程序模块；单元测试结果由设计人员进行核实和验证。

（2）面向对象的集成测试。设计人员确认移交产品各项功能完备和稳定性之后，进行集成测试；集成测试结果提交需求人员进行核实和验证。

（3）面向对象的系统测试。需求人员确认移交产品各项功能完备和稳定性之后，进行系统测试；系统测试结果经产品/项目经理核实和验证后，提交测试部进行测试。

（4）测试部进行 α 测试。

（5）市场中心组织用户进行 β 测试。

（6）递交产品到管理部或者最终用户。

说明：如果各项测试未通过或验证不合格，重复上一步工作。

面向对象软件测试的整体流程如图 2.12 所示。

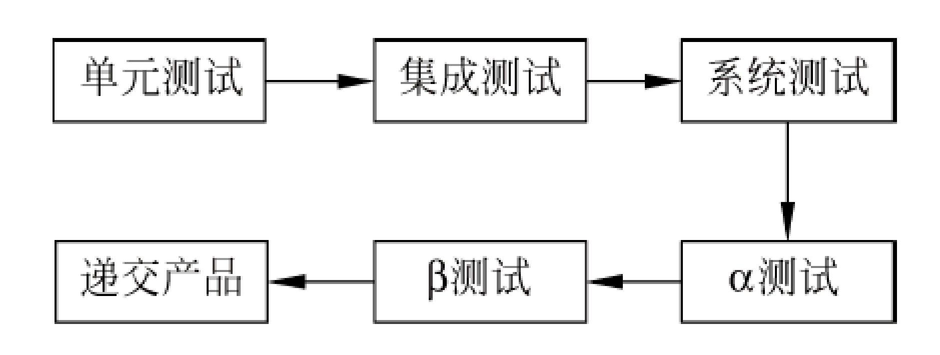


图 2.12　面向对象测试整体流程图

2.4.2　能力目标

了解面向对象技术的 6 个核心概念；掌握面向对象测试模型；掌握面向对象软件测试的整体流程。

2.4.3　任务驱动

1. 面向对象中类和对象之间的关系是什么？
2. 面向对象的开发模型实质是将软件测试过程分成几个阶段，每个阶段的侧重点是什么？
3. 面向对象单元测试中的测试单元是什么？
4. 面向对象集成测试分为几种？
5. 面向对象系统测试包括哪些方面？

2.4.4　实践环节

1. 使用 Java 语言针对 ATM 提款机终端系统进行面向对象的分析和编写程序。ATM 提款机操作的主要步骤如下。

(1) 输入密码，系统判断是否正确，如正确，进入取款界面；如错误，提示重新输入。

(2) 进入取款界面，输入取款金额，系统判断余额是否足够，如不足，提示；如足够，点钞。

(3) 吐出钞票，打印票据。

2. 对编写的 ATM 提款机系统进行面向对象的单元测试。

2.5　小　　结

- 软件测试在业界通常按照以下 4 种方式进行分类：按照开发阶段划分、按照测试方法或针对系统内部结构与具体实现算法划分、按照测试实施的组织划分和按照被测软件是否运行划分。
- 按照开发阶段划分为：单元测试、集成测试、系统测试、确认测试、验收测试、回归测试。
- 按照测试方法划分或针对系统内部结构与具体实现算法可以划分为：黑盒测试、白盒测试和灰盒测试。
- 按照测试实施的组织划分：开发方测试、用户测试、第三方测试。

- 按照被测软件是否运行划分：静态测试、动态测试。
- 常见的软件测试过程模型有 3 种：V 模型、W 模型和 H 模型。
- 软件测试工作的流程大体包括 5 个部分，分别是测试需求分析、制定测试计划、设计测试用例、测试用例执行、进行测试评估。
- 面向对象技术主要包括 6 个核心概念：对象、消息、接口、类、继承、多态。
- 面向对象测试的类型分为：面向对象分析的测试、面向对象设计的测试、面向对象编程的测试、面向对象单元测试、面向对象集成测试、面向对象系统测试。

习　题　2

1. 什么是静态测试、动态测试、黑盒测试、白盒测试？
2. 如果开发时间紧迫，是否可以跳过单元测试而直接进行集成测试？试说明原因。
3. 什么是驱动模块和桩模块？
4. 集成测试常用的集成策略有哪些？分别说明。
5. 软件测试流程包括几个步骤？
6. 分析比较面向对象的软件测试与传统的软件测试的异同。
7. 软件测试用例主要由输入数据和(　　)两部分组成。
   A. 测试计划　　B. 测试规则
   C. 预期输出结果　　D. 以往测试记录分析
8. 测试过程的 4 项基本活动是测试计划、测试设计、测试执行和(　　)。
   A. 测试运行　　B. 测试报告
   C. 测试度量　　D. 测试需求
9. 软件测试 V 模型中和概要设计阶段对应的测试是(　　)。
   A. 单元测试　　B. 集成测试
   C. 系统测试　　D. 验收测试
10. 下面说法正确的是(　　)。
    A. 软件测试是一个贯穿软件开发生命周期的活动
    B. 软件测试只在编码后进行
    C. 测试过程中应重视测试的执行，可以轻视测试的设计
    D. 因为测试工作简单，对软件产品质量影响不大
11. (　　)模型强调了测试计划等工作的先行与对系统需求与系统设计的测试。
    A. V 模型　　B. W 模型
    C. 渐进模型　　D. 螺旋模型
12. (　　)模型对软件测试流程予以了说明。
    A. V 模型　　B. W 模型
    C. H 模型　　D. 增量模型
13. 关于对第三方测试的描述，正确的观点是(　　)。
    A. 既不是开发人员，也不是用户所进行的测试就是第三方测试
    B. 由在技术、管理和财务上与开发方和用户方相对独立的组织进行的测试

C. 第三方测试是在开发方与用户方的测试基础上所进行的验证测试

D. 第三方测试又被称为β测试

14. 软件测试的手段有多种,通过人工来评审文档或程序,借以发现其中的错误,该手段是(　　)。

A. 黑盒测试　　B. 正确性测试

C. 动态测试　　D. 静态测试

15. 软件测试中根据测试用例设计的方法的不同可分为黑盒测试和白盒测试两种,它们(　　)。

A. 前者属于静态测试,后者属于动态测试

B. 都属于静态测试

C. 前者属于动态测试,后者属于静态测试

D. 都属于动态测试

16. 设计测试用例时,应当包括(　　)。

A. 合理的输入条件　　B. 不合理的输入条件

C. 合理的和不合理的输入条件　　D. 部分条件

17. 可以作为软件测试结束标志的是(　　)。

A. 使用了特定的测试用例

B. 错误强度曲线下降到预定的水平

C. 查出了预定数目的错误

D. 按照测试计划中所规定的时间进行了测试

18. 下面①～④是关于软件评测师工作原则的描述,正确的判断是(　　)。

① 对于开发人员提交的程序必须进行完全的测试,以确保程序的质量

② 必须合理安排测试任务,做好周密的测试计划,平均分配软件各个模块的测试时间

③ 在测试之前需要与开发人员进行详细的交流,明确开发人员的程序设计思路,并以此为依据开展软件测试工作,最大限度地发现程序中与其设计思路不一致的错误

④ 要对自己发现的问题负责,确保每一个问题都能被开发人员理解和修改

A. ①、②　　B. ②、③

C. ①、③　　D. 无

19. 软件测试是软件质量保证的重要手段,下述(　　)测试是软件测试的最基础环节。

A. 功能　　B. 单元

C. 结构　　D. 确认

20. 在进行单元测试的过程中,通常测试工程师都需要借助(　　)来代替所测模块调用的子模块。

A. 桩模块　　B. 驱动模块

C. 桩模块和驱动模块　　D. 存根模块和驱动模块

21. 在单元测试的基础上,需要将所有模块按照概要设计和详细设计说明书的要求进行组装,模块组装成系统的方式有两种,分别是(　　)。

A. 一次性组装和增殖性组装　　B. 自顶向下组装和自底向上组装

C. 单个模块组装和混合模块组装　　D. 接口组装和功能组装

22. 软件的集成测试工作最好由(　　)承担,以提高集成测试的效果。

A. 该软件的设计人员　　B. 该软件开发组的负责人

C. 该软件的编程人员　　D. 不属于该软件开发组的软件设计人员

23. V 模型指出(　　)测试对系统设计进行验证。

A. 单元　　B. 集成

C. 功能　　D. 系统

24. 对于软件的 β 测试,下列描述正确的是(　　)。

A. β 测试就是在软件公司内部展开的测试,由公司专业的测试人员执行的测试

B. β 测试就是在软件公司内部展开的测试,由公司的非专业测试人员执行的测试

C. β 测试就是在软件公司外部展开的测试,由专业的测试人员执行的测试

D. β 测试就是在软件公司外部展开的测试,可以由非专业的测试人员执行的测试

25. (　　)测试应当追溯到用户需求说明。

A. 代码　　B. 集成

C. 验收　　D. 单元

26. 验收测试的定义是(　　)。

A. 由用户按照用户手册对软件进行测试以决定是否接收

B. 由某个测试机构代表用户按照需求说明书和用户手册对软件进行测试以决定是否接收

C. 按照软件任务书或合同,供需双方约定的验收依据进行测试,决定是否接收

D. 由开发方和用户按照用户手册执行软件验收

第 3 章

黑 盒 测 试

主要内容

- 黑盒测试概述
- 等价类划分法
- 边界值分析法
- 决策表分析法

在本章中主要掌握黑盒测试的基本原理；掌握黑盒测试用例设计的主要方法：等价类划分法、边界值分析法、决策表分析法等，并灵活应用在实际的测试工作中。

3.1 黑盒测试概述

3.1.1 核心知识

1. 黑盒测试概念

黑盒测试(Black Box Testing)指的是把测试对象看做一个黑盒子，测试人员完全不考虑程序内部结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合要求。黑盒测试又叫做功能测试或数据驱动测试。

黑盒测试主要是通过将软件功能进行分解，然后再按照不同方法来设计测试用例。功能分解是把软件分解为相对独立的功能单元，其目的是通过功能分解可以明确软件功能测试的内容，使软件功能测试可以度量，有利于测试的监督和管理。功能分解应把握好度，不能分解得过粗，也不能分解得过细，最好按照功能的需求程度进行分解，要求高的软件应该分解得细一点，要求低的软件，测试要求可以粗略一点。

通过黑盒测试可以检测每个功能是否都能正常运行，因此黑盒测试又称为从用户观点和需求出发进行的测试。由于黑盒测试不考虑程序内部结构，只关心软件的功能，所以许多高层的测试(如确认测试、系统测试、验收测试)都主要采用黑盒测试。设计黑盒测试用例可以和软件实现同时进行，因此可以缩短整个测试的时间。

黑盒测试主要是为了发现以下错误：

(1) 待测系统是否有不正确或遗漏了的功能；

(2) 数据的输入能否正确地接收；

(3) 能否输出正确的结果；

(4) 是否有数据结构错误或外部信息(例如数据文件)访问错误。

2. 黑盒测试的优缺点

黑盒测试的主要优点如下：

(1) 从产品功能角度测试可以最大限度地满足用户的需求；

(2) 相同动作可重复执行，最枯燥的部分可由机器完成，容易实现自动化测试；

(3) 依据测试用例有针对性地寻找问题，定位更为准确，容易生成测试数据；

(4) 将测试直接和待测程序或系统要完成的操作相关联。

黑盒测试的主要缺点如下：

(1) 代码得不到测试；

(2) 如果需求规格说明设计有误，很难发现错误所在；

(3) 测试不能充分地进行；

(4) 结果的准确性取决于测试用例的设计。

由于黑盒测试只关心软件的外部功能和界面表现，不接触代码，为了保证测试工作顺利进行，应在合理的时间内完成测试工作，发现软件系统的缺陷，在对黑盒测试人员的选择和要求上也要符合一定的标准：通常要求掌握软件测试的基本思想和常规测试流程，了解产品的需求和功能，掌握测试用例的书写，有一定的软件开发和测试经验等。

黑盒测试对于整个测试工作有重要的意义：

(1) 黑盒测试有助于对被测产品的总体功能的需求进行验证；

(2) 从测试管理方面来说，黑盒测试是非常方便的，不需要对代码进行测试管理；

(3) 黑盒测试是把所有可能的输入都作为测试数据使用的，容易查出程序中的错误。

3. 黑盒测试用例设计的主要方法

黑盒测试用例设计主要分为等价类划分法、边界值分析法、决策表分析法、因果图法、错误推算法等。在本章中主要掌握等价类划分法、边界值分析法、决策表分析法。下面对主要方法进行概述。

(1) 等价类划分法

等价类划分法是黑盒测试用例设计中一种常用的设计方法，它将不能穷举的测试过程进行合理分类，从而保证设计出来的测试用例具有完整性和代表性。在划分等价类的过程中，不但要考虑有效等价类的划分，同时也要考虑无效等价类的划分。

- 有效等价类。有效等价类是指对软件需求规格说明来说，合理、有意义的输入数据所构成的集合。
- 无效等价类。无效等价类则和有效等价类相反，即不满足程序输入要求或者无效的输入数据所构成的集合。

(2) 边界值分析法

边界值分析法是一种补充等价类划分法的测试用例设计技术，它不是选择等价类中的任意元素，而是选择等价类边界的元素形成测试数据。在测试过程中，测试人员往往容易忽略边界值的条件，实践证明大量的错误都是发生在输入、输出范围的边界上，而不是发生在输入、输出范围的内部。因此针对各种边界情况设计测试用例，可以检查出更多的

错误。

(3) 决策表分析法

决策表分析法是分析和表达多逻辑条件下执行不同操作情况的工具。能够将复杂的问题按照各种可能的情况全部列举出来,从而避免遗漏。因此,利用决策表分析法能够设计出完整的测试用例集合。

(4) 因果图法

等价类划分法和边界值分析方法都是着重考虑输入条件,但没有考虑输入条件的各种组合、输入条件之间的相互制约关系。这样虽然各种输入条件可能出错的情况已经测试到了,但多个输入条件组合起来可能出错的情况却被忽视了。如果在测试时必须考虑输入条件的各种组合,则可能的组合数目将是天文数字,因此必须考虑采用一种适合于描述多种条件的组合、相应产生多个动作的形式来进行测试用例的设计,这就需要利用因果图法。

(5) 错误推算法

基于经验和直觉推测程序中所有可能存在的各种错误,从而有针对性的设计测试用例的方法。

3.1.2　能力目标

掌握黑盒测试的基本概念;掌握黑盒测试的优、缺点;了解黑盒测试用例设计的主要方法。

3.1.3　任务驱动

1. 试分析黑盒测试属于静态测试还是动态测试?请说明原因。
2. 黑盒测试人员是否需要编程经验?
3. 在测试的几个阶段(单元测试、集成测试、验收测试、确认测试)中,哪些以黑盒测试为主?试说明原因。

3.1.4　实践环节

1. 通过招聘网站,了解黑盒测试工程师的基本岗位要求、薪资待遇。
2. 分组讨论黑盒测试常用的测试方法的优、缺点。

3.2　等价类划分法

3.2.1　核心知识

等价类划分法是一种重要的、常用的黑盒测试方法,它将不能穷举的测试过程进行合理分类,从而保证设计出来的测试用例具有完整性和代表性。

1. 等价区间

等价区间是指将被测对象的输入和输出范围划分成一些区间,被测软件对一个特定区间的任何值都是等价的。形成测试区间的数据可以是函数或过程的参数,也可以是程序可

访问的全局变量、系统资源等，这些变量或资源可以是以时间形式存在的数据，或以状态形式存在的输入或输出序列。

等价类划分法的原理是把所有可能的输入数据，即程序的输入域划分成若干等价区间（子集），然后从每一个子集中选取少量具有代表性的数据作为测试用例。

通常设计测试用例时，在需求说明的基础上首先划分等价区间；其次在等价区间中选取等价类，列出等价类表；最后确定测试用例。

2. 等价类的分类

等价类按照其有效性可以分为两种：有效等价类和无效等价类。

(1) 有效等价类。对需求规格说明而言，有意义、合理的输入数据所组成的集合。用来检验程序是否实现了需求规格说明书中预先规定的功能和性能。

(2) 无效等价类。对需求规格说明而言，无意义的、不合理的输入数据所组成的集合。用来检查被测程序的功能和性能是否有不符合需求规格说明书中要求的地方。

3. 如何划分等价类

先从程序的需求规格说明书中找出各个输入条件，再为每个输入条件划分等价类，形成若干互不相交的子集。常见的划分原则如下。

(1) 如果规定了输入条件的取值范围或者个数，则可以确定一个有效等价类和两个无效等价类。

(2) 如果规定了输入值的集合，则可以确定一个有效等价类和一个无效等价类。

(3) 如果规定了输入数据的一组值，并且程序要对每一个输入值分别进行处理，则可为每一个值确定一个有效等价类，此外根据这组值确定一个无效等价类，即所有不允许的输入值的集合。

(4) 如果规定了输入数据必须遵守的规则，则可以确定一个有效等价类和若干个无效等价类。

(5) 如果已知的等价类中各个元素在程序中的处理方式不同，则应将该等价类进一步划分成更小的等价类。

4. 测试用例的设计

等价类划分法设计测试用例的步骤如下。

(1) 分析并确定等价类。

(2) 建立等价类表(或等价类图)，列出所有划分出的等价类。

(3) 从划分出的等价类中按以下 3 个原则设计测试用例。

- 为每一个等价类规定一个唯一的编号。
- 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止。
- 设计一个新的测试用例，使其仅覆盖一个尚未被覆盖的无效等价类，重复这一步，直到所有的无效等价类都被覆盖为止。

案例 1：测试学生成绩管理系统中的学生成绩录入模块，要求成绩的取值范围在 0～100 之间的整数。

在本例中如果使用黑盒测试方法测试该模块时，如果采用穷举测试的方式，可以采用的

测试数据会有很多，只要是整数值就可以，如 1、4、32、－68、103 等。测试人员如果不采用相关的测试方法减少测试数据数量的话，将造成测试时间的浪费，并且测试效果不明显，间接地提高了测试的成本。

采用黑盒测试中的等价类划分法，可以大量地减少测试数据的数量。先根据取值范围确定有效等价类和无效等价类。按照上述划分等价类的原则，本例中成绩的取值范围在 0～100 之间的整数，所以可以划分一个有效等价类和两个无效等价类。等价类图如图 3.1 所示。

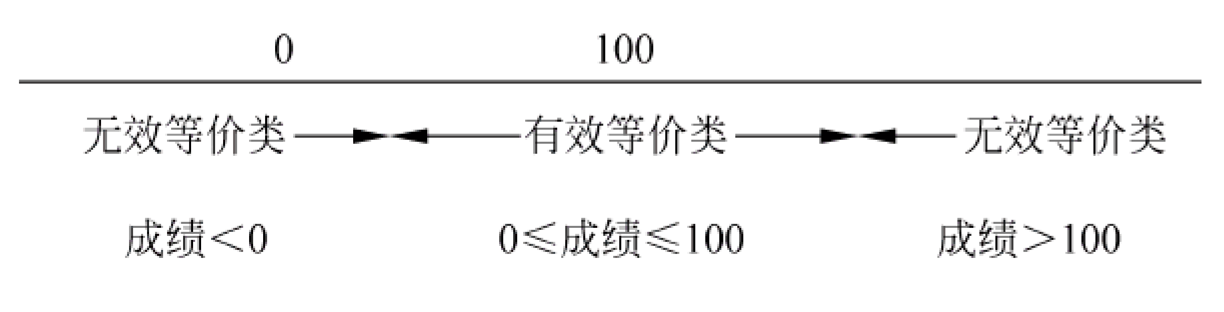


图 3.1　成绩取值范围等价类图

划分等价类后，假设每个等价类只包括一个等价区间，则分别在各个等价类的等价区间中选取一组测试数据就可以完成对该模块的测试，如－9、78、132。有效等价类的数据应该能正确录入到学生成绩中，并提示“成绩录入成功”。无效等价类的数据录入时，系统应该做相应的错误验证，提示“输入成绩有误”。最后形成的等价类测试如表 3.1 所示。

表 3.1　成绩取值范围等价类测试

测试模块	测试方法	测试区间	测试用例编号	预期输入	预期输出
成绩录入模块	有效等价类	0～100	1	78	成绩录入成功
	无效等价类	<0	2	－9	输入成绩有误
		>100	3	132	输入成绩有误

5. 其他常见等价类划分形式

根据是否对无效数据进行测试，可以将等价类测试分为两种：标准等价类和健壮等价类。

(1) 标准等价类

标准等价类不考虑无效数据值，测试用例使用每个等价类中的一个值；通常，标准等价类测试用例的数量和有效等价类中元素的数目相等。

标准等价类按照其等价区间的覆盖程度，又可以分为弱标准等价类和强标准等价类。

① 弱标准等价类。对有效等价区间进行划分，形成若干个等价区间，每个等价区间不相交，所有等价区间的并为整个有效等价区间。每个等价区间作为一个标准等价类出现，从中选取一组测试数据代表整个等价区间中的其他测试数据，要求测试数据覆盖整个有效范围。

案例 2：假设某待测模块要求输入两个数据，分别用两个变量 x1、x2 接受输入的值。要求两个输入变量的有效取值范围如下：

$$a \leqslant x1 \leqslant d,\quad 区间为[a,b],\ (b,c),\ [c,d]$$

$$e \leqslant x2 \leqslant g,\quad 区间为[e,f),\ [f,g]$$

则无效范围为：

x1＜a，x1＞d；x2＜e，x2＞g

弱标准等价类如图 3.2 所示。

说明：图中深色区域代表两个变量共同的有效取值范围。根据每个变量划分的等价区间，把深色区域细化为六小部分(即六个标准等价类)。根据弱标准等价类的划分原则，分别选取三个黑点所对应的 x1 和 x2 的值作为测试数据。这三组数据即覆盖了 x1 的三个有效区间，又覆盖了 x2 的两个有效区间。

② 强标准等价类。每一个标准等价类中要至少选择一个测试用例，测试数据覆盖整个有效区间。强标准等价类图如图 3.3 所示。

说明：六个黑点所对应的测试数据覆盖了整个有效区间。

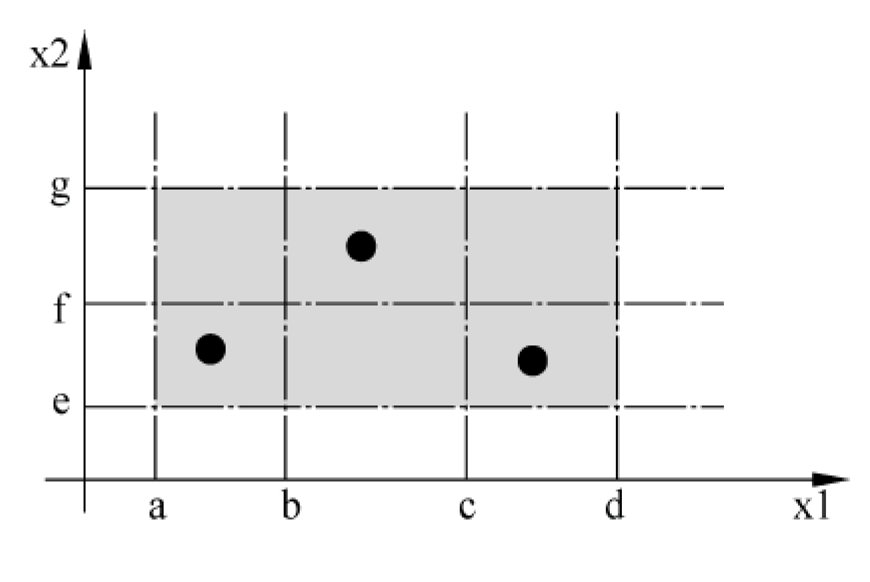


图 3.2　弱标准等价类图

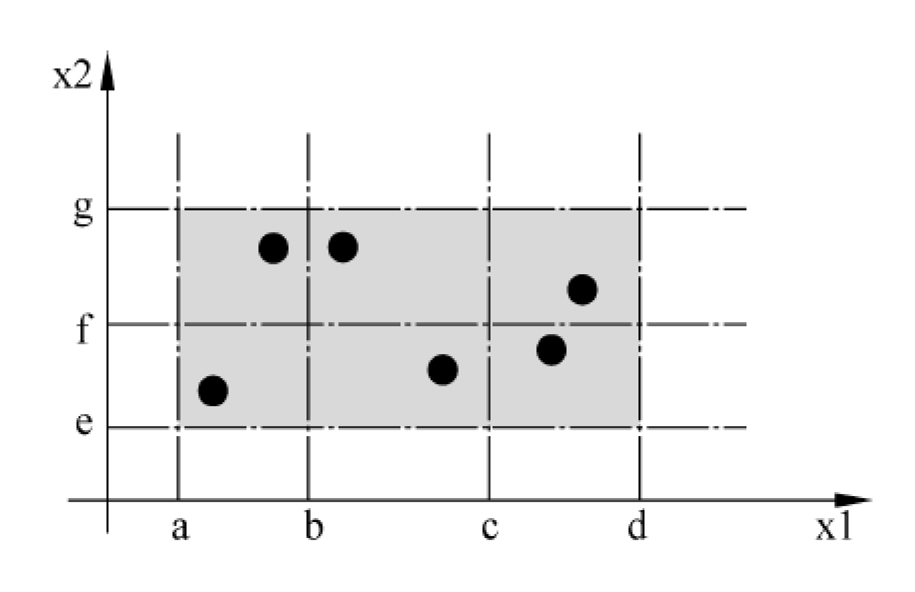


图 3.3　强标准等价类图

(2) 健壮等价类

健壮等价类对有效输入，测试用例从每个有效等价类中取一个值；对无效输入，一个测试用例有一个无效值，其他值均取有效值；通常，需求规格说明往往没有定义无效测试用例的期望输出，因此需要定义这些测试用例的期望输出。

健壮等价类按照其等价区间的覆盖程度，又可以分为弱健壮等价类和强健壮等价类。

① 弱健壮等价类。对于有效输入，使用每个有效类的一个值；对于无效输入，使用一个无效值，并保持其余的值都有效。弱健壮等价类图如图 3.4 所示。

说明：在深色区域中的三个黑点所对应的测试数据，覆盖了 x1 的三个有效区间，又覆盖了x2 的两个有效区间；同时四个浅色点在保证只有一个变量的取值为无效值，其余的变量取值为有效值。

② 强健壮等价类。每个有效等价类和无效等价类都至少要选择一个测试用例。强健壮等价类图如图 3.5 所示。

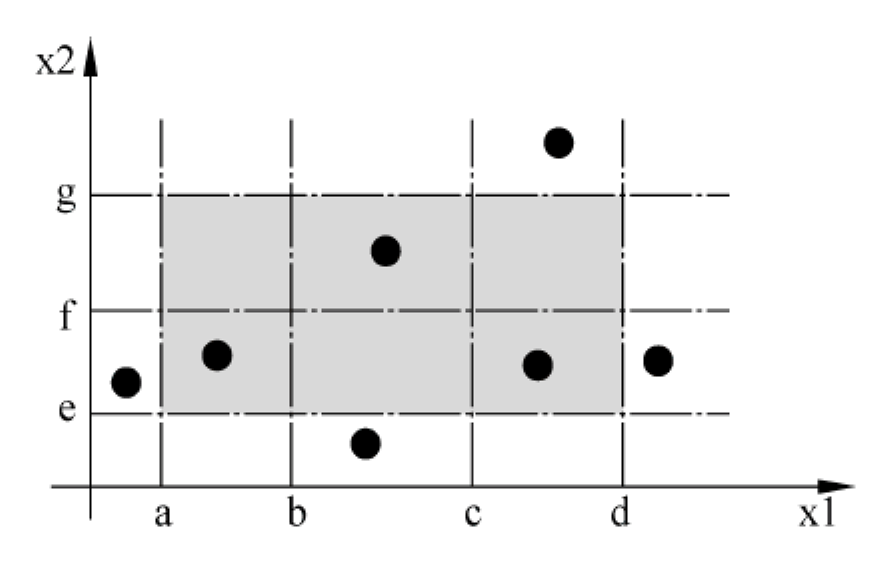


图 3.4　弱健壮等价类图

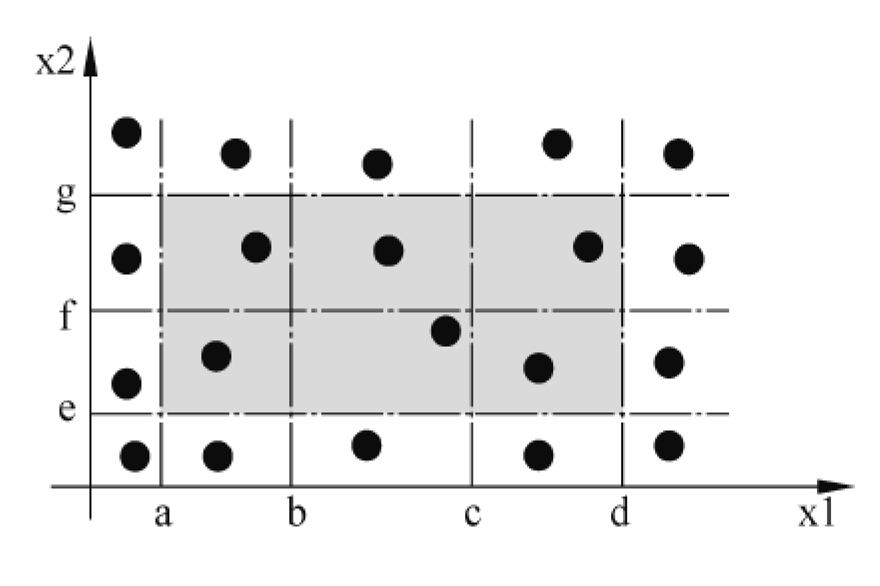


图 3.5　强健壮等价类图

3.2.2 能力目标

了解等价类划分法的基本思想；掌握等价类划分法的基本原则；掌握等价类划分法设计测试用例的步骤，能够设计测试用例。

3.2.3 任务驱动

1. 等价类划分法的基本原理是什么？

2. 使用等价类划分法设计测试用例来判断三个整数是否构成三角形。要求输入三个整数 a、b、c，分别作为三角形的三条边，取值范围在 1～100 之间，判断由三条边构成的三角形类型为等边三角形、等腰三角形、一般三角形以及不构成三角形。

3.2.4 实践环节

1. 有一个平方根函数要求当输入值为 0 或大于 0 时，返回输入数的平方根；当输入值小于 0 时，显示错误信息“平方根错误，输入值小于 0”，并返回 0。使用等价类划分法进行测试用例的设计。

2. 在注册人人网站时，要求用户必须输入用户名、密码及确认密码，对每一项输入条件的要求如下：用户名要求为 5 位以上，20 位以下，使用英文字母、数字、“_”，并且首字符必须为字母；密码要求为 6～16 位之间，只能使用英文字母、数字，并且区分大小写。使用等价类划分法进行测试用例的设计。

3.3 边界值分析法

3.3.1 核心知识

边界值分析法是对输入的边界值进行测试的一种黑盒测试方法。通常边界值分析法是作为对等价类划分法的补充，这种情况下，其测试用例来自等价类的边界。

测试实践表明，大量的故障往往发生在输入定义域的边界上，而不是在其内部。因此，针对各种边界情况设计测试用例，通常会取得很好的测试效果。

1. 边界值分析法与等价类划分法的区别

（1）边界值分析法不是从某个等价类中随便挑一个元素作为测试数据，而是使用该等价类的每个边界值作为测试数据出现。

（2）边界值分析不仅考虑输入条件，还要考虑输出空间产生的测试情况。

2. 如何用边界值分析法设计测试用例

（1）确定边界情况。

（2）选取正好等于、刚刚大于或刚刚小于边界，以及一个正常的值作为测试数据。

3. 边界值分析法的分类

按照测试数据的有效性，可以将边界值分析法分为：标准边界值测试和健壮边界值测试。

(1) 标准边界值测试只考虑有效数据范围内的边界值。对于一个有 n 个变量的程序,标准边界值分析测试程序会产生 4n+1 个测试用例。

(2) 健壮边界值测试会考虑有效和无效数据范围内的边界值。对于一个有 n 个变量的程序,健壮边界值分析测试程序会产生 6n+1 个测试用例。

通常情况下,软件测试所包含的边界检验有几种类型:数字、字符、位置、重量、大小、速度、方位、尺寸、空间等。相应的,以上类型的边界值应该在:最大(最小)、首位(末位)、上(下)、最快(最慢)、最高(最低)、最短(最长)、空(满)等情况下利用边界值作为测试数据。

4. 边界值分析法测试用例的设计原则

(1) 如果输入条件规定了值的范围,则应该取刚达到这个范围的边界值,以及刚刚超越这个范围边界的值作为测试输入数据。

(2) 如果输入条件规定了值的个数,则用最大个数、最小个数、比最小个数少一、比最大个数多一、正常值的数作为测试数据。

(3) 将规则(1)和(2)应用于输出条件,即设计测试用例使输出值达到边界值及其左右的值。

(4) 如果程序的规格说明给出的输入域或输出域是有序集合,则应选取集合的第一个元素和最后一个元素作为测试用例。

(5) 如果程序中使用了一个内部数据结构,则应当选择这个内部数据结构的边界上的值作为测试用例。

(6) 分析需求规格说明书,找出其他可能的边界条件。

案例 3:使用边界值分析法对学生成绩管理系统中的学生成绩录入模块设计测试用例。

本例中成绩的取值范围在 0~100 之间的整数。成绩的边界为 0 或者 100。

按照标准边界值分析所设计出的一组测试数据如表 3.2 所示。

表 3.2　成绩标准边界值测试用例表

测试模块	测试方法	测试区间	测试用例编号	预期输入	预期输出
成绩录入模块	边界值分析法	等于 0	1	0	成绩录入成功
		略大于 0	2	1	成绩录入成功
		0~100	3	60	成绩录入成功
		略小于 100	4	99	成绩录入成功
		等于 100	5	100	成绩录入成功

按照健壮边界值分析所设计出的一组测试数据如表 3.3 所示。

表 3.3　成绩健壮边界值测试用例表

测试模块	测试方法	测试区间	测试用例编号	预期输入	预期输出
成绩录入模块	边界值分析法	略小于 0	1	−1	无效成绩
		等于 0	2	0	成绩录入成功
		略大于 0	3	1	成绩录入成功
		0~100	4	60	成绩录入成功
		略小于 100	5	99	成绩录入成功
		等于 100	6	100	成绩录入成功
		略大于 100	7	101	无效成绩

说明：从两张测试用例表中可以看出，对于本例中只有一个输入数据，分别得到 5 个和 7 个测试用例，满足 4n+1 和 6n+1 的设计规律。

案例 4：使用边界值分析法测试一个函数 Test(int x,int y)，该函数有两个变量 x 和 y，x、y 的取值范围分别是：5≤x≤20，5≤y≤15。

标准边界值测试用例图如图 3.6 所示。

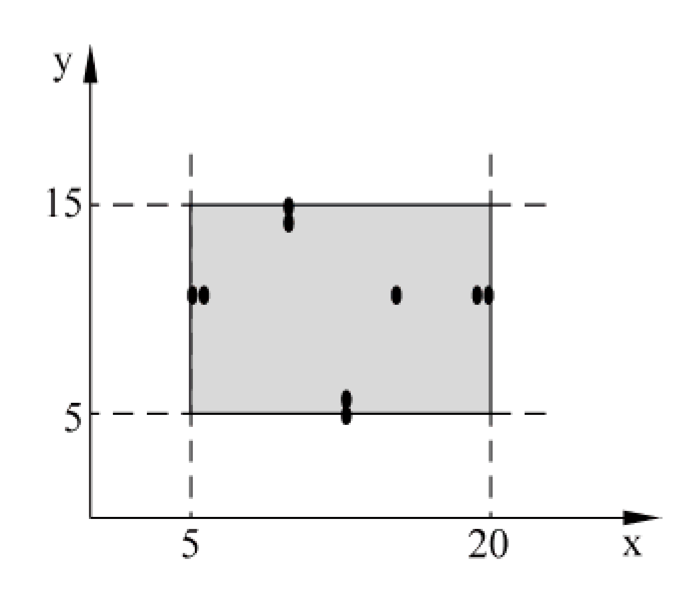


图 3.6　标准边界值测试用例图

根据上图分析设计出的测试用例表如表 3.4 所示。

表 3.4　标准边界值测试用例表

测试模块	测试方法	测试区间	测试用例编号	预期输入	预期输出
Test(int x,int y)	边界值分析法	x 为最小值，y 为正常值	1	x=5，y=10	有效
		x 略大于最小值，y 为正常值	2	x=6，y=10	有效
		x 略小于最大值，y 为正常值	3	x=19，y=10	有效
		x 等于最大值，y 为正常值	4	x=20，y=10	有效
		x 为正常值，y 为最小值	5	x=15，y=5	有效
		x 为正常值，y 略大于最小值	6	x=15，y=6	有效
		x 为正常值，y 略小于最大值	7	x=15，y=14	有效
		x 为正常值，y 等于最大值	8	x=15，y=15	有效
		x 为正常值，y 为正常值	9	x=17，y=10	有效

健壮边界值测试用例图如图 3.7 所示。

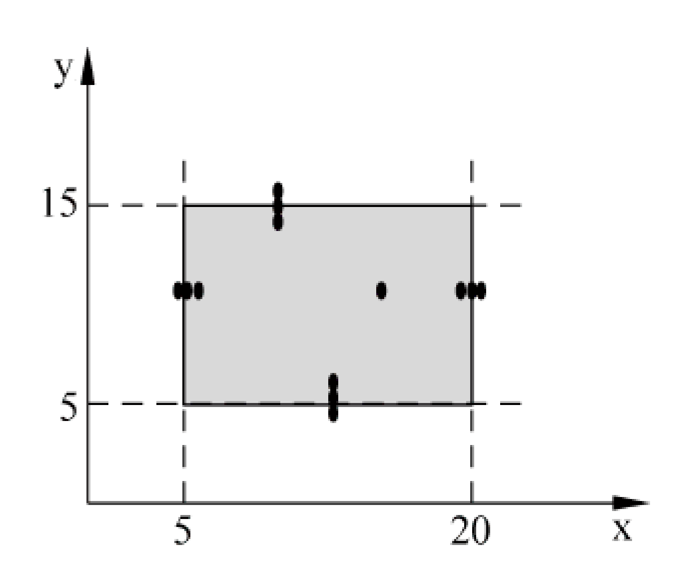


图 3.7　健壮边界值测试用例图

根据上图分析设计出的测试用例表如表 3.5 所示。

表 3.5　健壮边界值测试用例表

测试模块	测试方法	测试区间	测试用例编号	预期输入	预期输出
Test(int x,int y)	边界值分析法	x 为最小值，y 为正常值	1	x=5，y=10	有效
		x 略大于最小值，y 为正常值	2	x=6，y=10	有效
		x 略小于最大值，y 为正常值	3	x=19，y=10	有效
		x 等于最大值，y 为正常值	4	x=20，y=10	有效
		x 为正常值，y 为最小值	5	x=15，y=5	有效
		x 为正常值，y 略大于最小值	6	x=15，y=6	有效
		x 为正常值，y 略小于最大值	7	x=15，y=14	有效
		x 为正常值，y 等于最大值	8	x=15，y=15	有效
		x 为正常值，y 为正常值	9	x=17，y=10	有效
		x 略小于最小值，y 为正常值	10	x=4，y=10	无效
		x 略大于最大值，y 为正常值	11	x=21，y=10	无效
		x 为正常值，y 略小于最小值	12	x=15，y=4	无效
		x 为正常值，y 略大于最大值	13	x=15，y=16	无效

说明：从两张测试用例表中可以看出，对于本例中有两个输入数据，分别得到 9 个和 13 个测试用例，满足 4n+1 和 6n+1 的设计规律。

3.3.2　能力目标

了解边界值分析法的基本思想；了解边界值分析法与等价类划分法的区别；掌握边界值分析法测试用例的设计原则；掌握边界值分析法测试用例的步骤，能够设计测试用例。

3.3.3　任务驱动

1. 标准边界值分析法与健壮边界值分析法的区别是什么？

2. 有一个计算平方根的函数 sqrt(int x)，x 的取值范围是大于等于 0 的所有整数，请用标准边界值分析法与健壮边界值分析法分别针对该函数设计测试用例。

3. 使用边界值分析法设计测试用例来判断三个整数是否构成三角形。要求输入三个整数 a、b、c，分别作为三角形的三条边，取值范围在 1～100 之间，判断由三条边构成的三角形类型为等边三角形、等腰三角形、一般三角形以及不构成三角形。

3.3.4　实践环节

1. 有一个信息管理系统，要求用户输入以年月表示的日期。假设日期限定在 1990 年 1 月至 2049 年 12 月，并规定日期由 6 位数字字符组成，前 4 位表示年，后 2 位表示月。使用边界值分析法设计测试用例，来测试程序的“日期检查功能”。

2. 某电商促销系统对在线购物商品进行促销打折活动，具体活动细节如下：

- 购物金额＞＝88 元，9 折优惠；
- 购物金额＞＝588 元，8 折优惠；
- 购物金额＞＝1088 元，7 折优惠。

使用边界值分析法设计测试用例。

3.4　决策表分析法

3.4.1　核心知识

在一些数据处理问题当中，某些操作的实施依赖于多个逻辑条件的组合，即：针对不同逻辑条件的组合值，分别执行不同的操作。决策表分析法就是分析和表达多逻辑条件下执行不同操作情况的黑盒测试方法。

1. 决策表的组成

决策表通常由四个部分组成：条件桩、条件项、动作桩、动作项。决策表组成图如图 3.8 所示。

(1) 条件桩：列出问题的所有条件；

(2) 条件项：针对条件桩给出的条件列出所有可能的取值；

(3) 动作桩：列出问题规定的可能采取的操作；

(4) 动作项：指出在条件项的各组取值情况下应采取的动作。

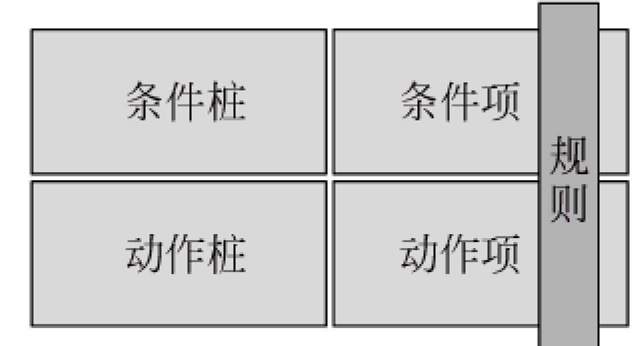


图 3.8　决策表组成图

2. 规则

任何一个条件组合的特定取值及其相应要执行的动作称为规则。

3. 决策表的产生规则

决策表的产生规则如下。

(1) 列出所有的条件桩和动作桩。

(2) 根据条件桩确定规则的个数。有 n 个条件的决策表有 2^n 个规则(每个条件取真、假值)。

(3) 填入条件项。

(4) 填入动作项,得到初始决策表。

(5) 简化决策表,合并相似规则。

若表中有两条以上规则具有相同的动作,并且在条件项之间存在极为相似的关系,便可以合并。合并后的条件项用符号"—"表示,说明执行的动作与该条件的取值无关,称为"无关条件"。

4. 适合使用决策表设计测试用例的情况

(1) 规格说明以判定表(即,给出条件进行判断)形式给出,或是很容易转换成判定表;

(2) 条件的排列顺序不会影响执行哪些动作;

(3) 规则的排列顺序不会影响执行哪些动作;

(4) 每当某一规则的条件已经满足,并确定要执行的动作后,不必检验别的规则;

(5) 如果某一规则得到满足要执行多个动作,这些动作的执行顺序无关紧要。

案例 5:某生产决策系统会根据上个月的销售情况以及库存情况进行生产的决策分析。如果某产品销售好并且库存低,则增加该产品的生产;如果该产品销售好,但库存量不低,则继续生产;若该产品销售不好,但库存量低,则继续生产;若该产品销售不好,且库存量不低,则停止生产。

根据题目可以知道决定继续生产的条件有两个:销售情况和库存情况。生产企业所能采取的动作有三个:增加生产、继续生产和停止生产。

销售情况可以有两个取值:好与不好。库存情况也有两个取值:好与不好。这里用 T 代表好,F 代表不好。由于只有两个条件,所以该决策表中共有 $2^2=4$ 个规则。

生产决策系统所对应的决策表如表 3.6 所示。

表 3.6　生产决策表

条件桩	条　件　项			
	规则 1	规则 2	规则 3	规则 4
C1:销售好?	T	T	F	F
C2:库存低?	T	F	T	F
动作桩	动　作　项			
A1:增加生产	√			
A2:继续生产		√	√	
A3:停止生产				√

说明：决策表最突出的优点是能够将复杂的问题按照各种可能的情况全部列举出来，简单明了并且可以避免遗漏。因此，利用决策表能够设计出完整的测试用例集合。运用决策表设计测试用例，可以将条件作为输入，将动作作为输出。

案例 6：使用决策表分析法来判断三个整数是否构成三角形（a、b、c 分别代表三个整数）。

三角形有一个很重要的定理就是“两边之和大于第三边”。从这个原理出发，可以得到三个条件：a＋b＞c、a＋c＞b、b＋c＞a。只要不满足其中的任意一个条件，都不是三角形。由于三角形有以下几种情况：不等边三角形、等腰三角形、等边三角形。它们的判断与三个边有直接的关系。如不等边三角形三条边都不相等，等腰三角形要求两边相等，等边三角形要求三条边都相等。从中可以得到判断三角具体情况的三个条件：a 是否等于 b，b 是否等于 c，c 是否等于 a。

分析得出了六个条件，所以该决策表中共有 $2^6=64$ 个规则。

经过分析，得到决策表中的动作桩有：不是三角形、不等边三角形、等腰三角形、等边三角形和不可能（不可能是由于 a＝b，b＝c，而 a！＝c 这种组合情况引起的，由于决策表是所有条件的任意组合，所以这种不可能发生的情况却发生了。决策表分析法可以进行完备测试）。

在决策表中分别填写条件桩、动作桩、条件项以及动作项，规则共 64 条。在填写的过程中，发现当某个条件不满足时，其他条件的就算取任意 T 或者 F，对动作项没有影响。决策表初始情况表如表 3.7 所示。

表 3.7　决策表初始情况表

条　件　桩	条　件　项							
	规则 1	规则 2	规则 3	规则 4	规则 5	规则 6	规则 7	…
C1：a＜b＋c？	F	F	F	F	F	F	F	…
C2：b＜a＋c？	F	T	F	F	F	F	T	…
C3：c＜a＋b？	F	F	T	F	F	F	T	…
C4：a＝b？	F	F	F	T	F	F	F	…
C5：a＝c？	F	F	F	F	T	F	F	…
C6：b＝c？	F	F	F	F	F	T	F	…
动　作　桩	动　作　项							
A1：非三角形	√	√	√	√	√	√	√	…
A2：不等边三角形								…
A3：等腰三角形								…
A4：等边三角形								…
A5：不可能								…

说明：当条件 C1：a＜b＋c 不满足时，剩下 5 个条件不起决定作用，所对应的动作项都为 A1：非三角形。这时候可以将相似规则进行合并。

规则合并后的决策表如表 3.8 所示。

表 3.8　规则合并后的决策表

条件桩	条件项										
	1	2	3	4	5	6	7	8	9	10	11
C1:a＜b＋c?	F	T	T	T	T	T	T	T	T	T	T
C2:b＜a＋c?	—	F	T	T	T	T	T	T	T	T	T
C3:c＜a＋b?	—	—	F	T	T	T	T	T	T	T	T
C4:a＝b?	—	—	—	T	T	T	T	F	F	F	F
C5:a＝c?	—	—	—	T	T	F	F	T	T	F	F
C6:b＝c?	—	—	—	T	F	T	F	T	F	T	F
动作桩	动作项										
A1:非三角形	√	√	√								
A2:不等边三角形											√
A3:等腰三角形							√		√	√	
A4:等边三角形				√							
A5:不可能					√	√		√			

说明：原本 64 条规则，经过合并后，得到有效规则 11 条。

针对这 11 条规则分别设计测试用例如表 3.9 所示。

表 3.9　决策表分析法测试用例表

测试模块	测试方法	测试用例编号	预期输入			预期输出
			a	b	c	
判断三个整数是否构成三角形	决策表分析法	1	4	1	2	非三角形
		2	1	4	2	非三角形
		3	1	2	4	非三角形
		4	5	5	5	等边三角形
		5	X	X	X	不可能
		6	X	X	X	不可能
		7	2	2	3	等腰三角形
		8	X	X	X	不可能
		9	2	3	2	等腰三角形
		10	3	2	2	等腰三角形
		11	3	4	5	不等边三角形

3.4.2　能力目标

了解决策表分析法的基本思想和基本组成；掌握决策表的产生规则；具备使用决策表分析法设计测试用例的能力。

3.4.3　任务驱动

1. 决策表分析法由几部分组成？
2. 假设中国某航空公司规定：中国去欧美的航线所有座位都有食物供应，每个座位都

可以播放电影；中国去非欧美的国外航线都有食物供应，只有商务舱可以播放电影；国内航班的商务舱有食物供应，但是不可以播放电影；国内航班的经济舱除非飞行时间大于2小时有食物供应，但是不可以播放电影。使用决策表法设计测试用例。

3.4.4　实践环节

1. 讨论决策表分析法中的规则在什么情况下可以合并。
2. 使用决策表分析法对自动售货机进行软件测试用例的设计。

说明如下：若投入 10 元的纸币，按下“橙汁”或“可乐”的按钮(饮料都为 3 元一罐)，则相应的饮料就送出来，并找零。若售货机没有零钱找，则显示“零钱找完”的红灯亮，这时再投入 10 元按下按钮，饮料不会送出，并退还 10 元钱。

3.5　小　　结

- 黑盒测试中测试人员完全不考虑程序内部结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合功能说明。黑盒测试又叫做功能测试或数据驱动测试。
- 黑盒测试用例设计主要分为等价类划分法、边界值分析法、决策表分析法、因果图法、错误推算法等。
- 等价类划分法的原理是把所有可能的输入数据，即程序的输入域划分成若干等价区间(子集)，然后从每一个子集中选取少量具有代表性的数据作为测试用例。等价类按照其有效性可以分为两种：有效等价类和无效等价类。根据是否对无效数据进行测试，可以将等价类测试分为两种：标准等价类和健壮等价类。
- 边界值分析法就是对输入的边界值进行测试的一种黑盒测试方法。按照测试数据的有效性，可以把边界类测试法分为：标准边界值测试和健壮边界值测试。标准边界值测试只考虑有效数据范围内的边界值。对于一个有 n 个变量的程序，标准边界值分析测试程序会产生 4n+1 个测试用例。健壮边界值测试会考虑有效和无效数据范围内的边界值。对于一个有 n 个变量的程序，健壮边界值分析测试程序会产生 6n+1 个测试用例。
- 决策表分析法就是分析和表达多逻辑条件下执行不同操作情况的黑盒测试方法。决策表通常由四个部分组成：条件桩、条件项、动作桩、动作项。

习　题　3

1. 简述黑盒测试方法的特点。
2. 标准等价类划分法和健壮等价类划分法的区别是什么？
3. 决策表分析法为什么可以进行完备测试？
4. 测试银行提款机上的提款功能，要求用户输入的提款金额的有效数值是 100～3000，并以 100 为最小单位(即取款金额为 100 的倍数)试用等价类划分法和边界值分析法设计测试用例。

5. 某程序要求输入日期，规定变量 month、day、year 的取值范围为：1≤month≤12，1≤day≤31，1958≤year≤2058，试用边界值分析法设计测试用例。

6. 黑盒测试是通过软件的外部表现来发现软件缺陷和错误的测试方法，具体地说，黑盒测试用例设计技术包括(　　)。

A. 等价类划分法、因果图法、边界值分析法、错误推测法、决策表分析法

B. 等价类划分法、因果图法、边界值分析法、正交试验法、符号法

C. 等价类划分法、因果图法、边界值分析法、功能图法、基本路径法

D. 等价类划分法、因果图法、边界值分析法、静态质量度量法、场景法

7. 下面(　　)不是黑盒测试的主要特点。

A. 不需要了解程序内部的代码及实现

B. 与软件的内部实现无关

C. 从用户角度出发能很容易地知道用户用到哪些功能

D. 在进行人工测试时较为方便

8. 若有一个计算类型的程序，它的输入量只有一个 X，其范围是[−1.0，1.0]，现从输入的角度考虑一组测试用例：−1.001，−1.0，1.0，1.001。设计这组测试用例的方法是(　　)。

A. 条件覆盖法　　B. 等价分类法

C. 边界值分析法　　D. 错误推测法

9. 下面(　　)方法能够有效地检测输入边界可能引起的错误。

A. 等价类划分　　B. 边界值分析

C. 错误推测　　D. 因果图

10. 用边界值分析法，假定 1<X<100，那么 X 在测试中应该取的边界值是(　　)。

A. X=1，X=100　　B. X=0，X=1，X=100，X=101

C. X=2，X=99　　D. X=0，X=101

11. 如果一个函数有 n 个变量，采用边界值的方法进行黑盒测试那么总共需要的测试用例是(　　)。

A. 5^n　　B. 4 * n +1

C. 2n　　D. 7^n

12. 在等价类测试中，下列对等级类的划分不正确的是(　　)。

A. 根据等价关系对输入或输出数据的集合进行划分

B. 将集合划分为互不相交的子集

C. 划分子集的并是整个集合

D. 集合可以划分为相交的子集

13. 在黑盒测试中，着重检查输入条件组合的方法是(　　)。

A. 等价类划分法　　B. 边界值分析法

C. 错误推测法　　D. 决策表法

14. 用黑盒法设计测试用例时，采用的方法包括(　　)。

A. 判定覆盖法　　B. 条件覆盖法

C. 决策表法　　D. 路径分析法

15. 决策表由四部分组成：左上部列出(　　)。

A. 条件组合与动作之间的对应关系　　B. 所有条件

C. 所有可能的动作　　D. 可能的条件组合

16. 决策表由四部分组成：下面(　　)部分不属于这四部分之一。

A. 条件桩　　B. 条件项

C. 动作项　　D. 结果桩

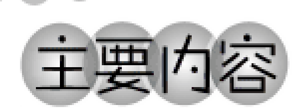
17. 错误推测法测试的主要依据是(　　)。

A. 条件项　　B. 等价类的划分

C. 测试数据的边界　　D. 测试人员的经验

第 4 章

白盒测试



- 白盒测试概述
- 逻辑覆盖法
- 独立路径测试法
- 面向对象的白盒测试

在本章中主要掌握白盒测试的基本原理；掌握白盒测试用例设计的主要方法：逻辑覆盖法、独立路径测试法、面向对象的白盒测试等，并灵活应用在实际的测试工作中。

4.1 白盒测试概述

4.1.1 核心知识

1. 白盒测试概念

白盒测试是测试者能够看到被测源程序，可以分析被测程序的内部结构，按照程序内部逻辑来测试程序，检查程序中每条通路是否按预定要求正确工作，是基于代码的测试，适用于开发领域按程序内部逻辑结构和编码结构设计测试数据的测试方法，对所有逻辑路径进行测试，通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致，又称为结构性测试或逻辑驱动测试。

2. 白盒测试分类

在测试过程中，常见的白盒测试技术主要包括：逻辑覆盖法、独立路径测试法、面向对象的白盒测试等。

(1) 逻辑覆盖法

逻辑覆盖是以程序的内部逻辑结构为基础的测试用例设计技术。它要求测试人员十分清楚程序的逻辑结构，考虑的是测试用例对程序内部逻辑覆盖的程度。

逻辑覆盖法又可以分为语句覆盖、判定覆盖、条件覆盖、判定条件覆盖、条件组合覆盖、路径覆盖。

① 语句覆盖指的是设计若干测试用例来执行程序代码中的语句，使被执行的语句数与所有可能的语句数达到一定的比例。例如：有些项目要求程序代码达到 100% 的语句

覆盖。

② 判定覆盖是一种针对判定结果设计测试用例的技术，是执行测试套件能够覆盖的判定结果的百分比，即被执行的判定和总的判定的比值。在低级别的测试中，判定覆盖常常可以作为测试出口准则之一，例如：测试出口准则可以要求测试对象达到 100％的判定覆盖。100％的判定覆盖可以保证 100％的语句覆盖。

③ 条件覆盖指的是设计若干测试用例来执行不同的条件结果，是执行测试套件能够覆盖到原子条件的百分比，即被执行的原子条件和总的原子条件的比值。100％的条件覆盖要求测试覆盖到每一个原子条件语句分别取值为真和假的情况。需要注意的是，条件覆盖并不一定比判定覆盖的覆盖能力更强。

④ 判定条件覆盖指的是设计若干测试用例来执行条件结果和判定结果，是执行测试用例套件能够覆盖的条件结果和判定结果的百分比。100％的判定条件覆盖意味着 100％的判定覆盖和 100％的条件覆盖。

⑤ 条件组合覆盖是指设计测试用例覆盖每条语句中的原子条件所有可能的取值结果组合(即每个判定中的所有可能的原子条件取值组合至少执行一次)，是测试套件覆盖每条语句内的所有原子条件取值结果组合的百分比。100％条件组合覆盖意味着 100％判断条件覆盖。

⑥ 路径覆盖是指设计测试用例用来执行不同的路径，是测试套件执行的路径占总的路径的百分比。100％的路径覆盖可以确保 100％的语句覆盖和判定覆盖。

根据测试对象的复杂程度和不同的覆盖率要求，可以选择和应用不同的测试设计方法。不同覆盖方式的覆盖率强度关系如图 4.1 所示。

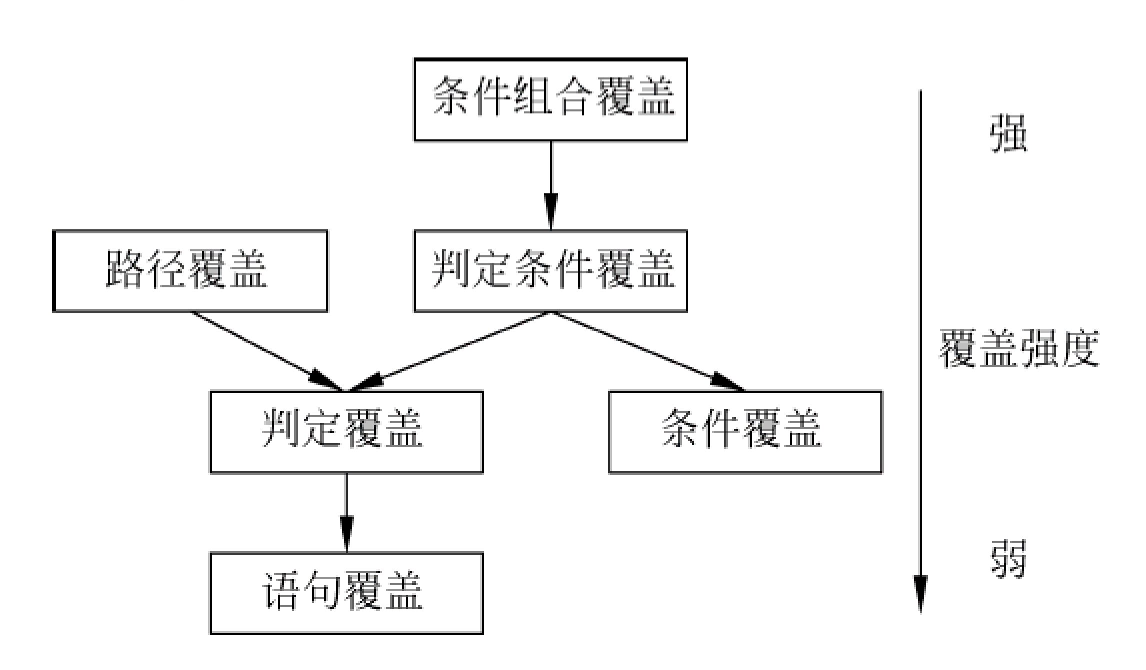


图 4.1　逻辑覆盖中各种覆盖率强度关系对比图

(2) 独立路径测试法

独立路径测试法是在程序控制流图的基础上，通过分析控制结构的环路复杂性，导出独立可执行路径集合，从而设计测试用例的方法。设计出的测试用例要保证在测试中程序的每个可执行路径至少执行一次。独立路径测试法包括以下 4 个步骤。

① 画出程序的控制流图：描述程序控制流的一种图示方法。

② 计算程序的环形复杂度。从程序的环形复杂度中可导出程序基本路径集合中的独立路径条数，这是确定程序中每个可执行语句至少执行一次所必需的测试用例数目的上界。

③ 导出测试用例。根据环形复杂度和程序结构设计测试用例，包括输入数据和预期结果。

④ 准备测试用例。确保独立路径集中的每一条路径都能够被执行。

(3) 面向对象的白盒测试

面向对象的白盒测试把类作为一个单元来进行测试。测试分为两层：第一层考虑类中各独立方法的代码；第二层考虑方法之间的相互作用。

面向对象软件的白盒测试主要是针对软件设计中的类和对象来进行测试的。因此了解被测试软件的类结构，是进行白盒测试的关键。由于封装的原则，在面向对象软件类中的成员一般设计为私有或受保护类型，即类的属性和方法是无法从外部直接访问的，必须通过类中的公有方法来实现。因此设计测试用例时必须注意对这些公有成员的测试。为了实现对类中所有方法的有效测试，必须设计足够多的测试用例。

4.1.2 能力目标

了解白盒测试的基本分类；掌握逻辑覆盖的几种测试方法；掌握独立路径测试的基本步骤；了解面向对象测试的基本思想。

4.1.3 任务驱动

1. 白盒测试的主要测试方法有哪些？
2. 分析归纳逻辑覆盖测试的六种覆盖策略的各自特点。
3. 简述独立路径测试的基本步骤。
4. 面向对象的白盒测试的侧重点是什么？

4.1.4 实践环节

1. 通过招聘网站，了解白盒测试工程师的基本岗位要求、薪资待遇。
2. "白盒测试工程师不需要有编程经验"这个命题是否正确请说明原因。
3. 路径覆盖和独立路径测试的区别是什么？

4.2 逻辑覆盖法

4.2.1 核心知识

逻辑覆盖是以程序的内部逻辑结构为基础的测试用例设计技术。它要求测试人员十分清楚程序的逻辑结构，考虑的是测试用例对程序内部逻辑覆盖的程度。

逻辑覆盖法又可以分为：语句覆盖、判定覆盖、条件覆盖、判定条件覆盖、条件组合覆盖、路径覆盖。

下面通过案例1来具体说明逻辑覆盖法中的六种覆盖方式。

案例1：某个程序中的流程图如图4.2所示，求程序运行后X的值。

使用逻辑覆盖法中六种覆盖方式分别设计测试用例。

为了能够更好地表达逻辑覆盖的含义，这里对条件和判断分别采用符号进行标示。

表示某个条件的真假使用 T_i 和 F_i(i为该条件的序号)，表示某个判定的真假使用 T_{Di} 和 F_{Di}，举例如下。

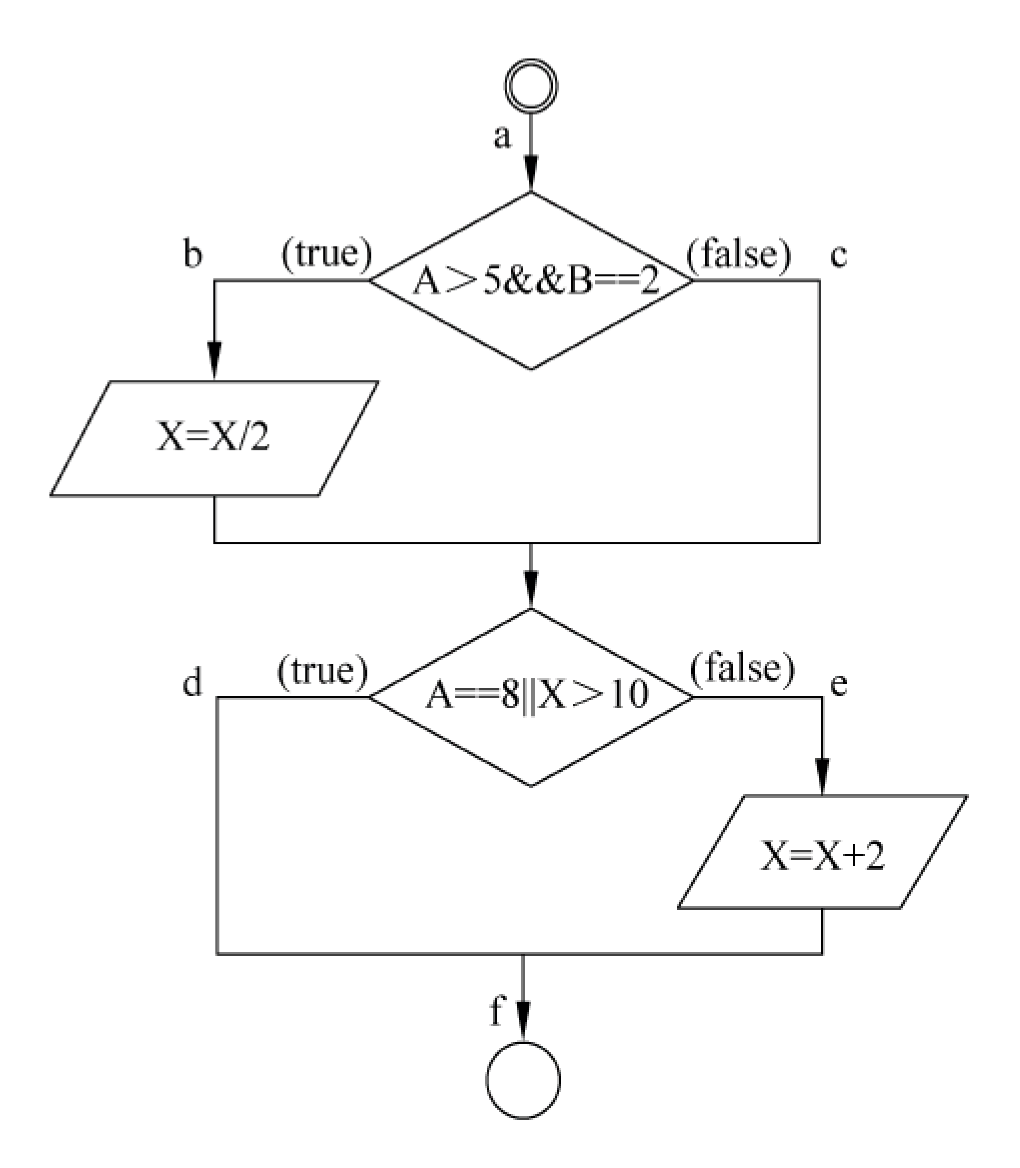


图 4.2　案例 1 程序流程图

条件：A>5 记为 T_1，A<=5 记为 F_1；

B==2 记为 T_2，B!=2 记为 F_2；

A==8 记为 T_3，A!=8 记为 F_3；

X>10 记为 T_4，X<=10 记为 F_4。

判定：A>5 && B==2 记为 T_{D1}，判定为假，记为 F_{D1}；

A==8||X>10 记为 T_{D2}，判定为假，记为 F_{D2}。

语句：表示程序中的独立语句，用 S_i 表示(i 代表语句的序号)。

例如：S_1：X=X/2；S_2：X=X+2

路径：表示程序从开始到结束的一条通路，用 R_i 表示(i 代表路径的序号)。

R_1：a→b→d→f；R_2：a→b→e→f；

R_3：a→c→d→f；R_4：a→c→e→f。

经过上述分析，本例中有两个可执行语句，四个条件，两个判定，四条路径。

(1) 语句覆盖

语句覆盖指的是设计若干测试用例来执行程序代码中的语句，要求每一条可执行语句至少执行一次，语句覆盖率是指被执行的语句数与所有可能的语句数之间的比值。

满足语句覆盖要求的测试用例如表 4.1 所示。

表 4.1　语句覆盖测试用例表

逻辑覆盖类型	条件	判定	测试用例编号	预期输入			预期输出
				A	B	X	
语句覆盖(S_1,S_2)	$T_1T_2F_3F_4$	$T_{D1}F_{D2}$	1	6	2	8	X=6

说明：该例中满足语句覆盖的测试用例不止一个，读者可以自行思考其他的输入值，覆盖两条语句。由于本例中只用两条语句，用一个测试用例就全部执行到了，所以语句覆盖率为100%。通常在测试工作中，要求语句覆盖率越高越好。但是语句覆盖也存在着致命的缺陷。比如下面代码所示。

```
if(x>5){
     statements;
...;(共 9999 条语句)
}else{
     statement;
}
```

设计语句覆盖测试用例时，可以指定 x=6，则 9999 条语句都可以执行，语句覆盖率为9999/10000*100%=99.99%。覆盖率已经很高了，但是有一个重要的分支没有覆盖。所以及时语句覆盖率达到100%，也不能代表对测试模块进行了完备测试。

(2) 判定覆盖

判定覆盖是一种针对判定结果设计测试用例的技术，要求每个判断的真分支和假分支至少经历一次。判断覆盖率是执行测试用例能够覆盖的判定结果的百分比，即被执行的判定和总的判定的比值。

满足判定覆盖要求的测试用例如表 4.2 所示。

表 4.2　判定覆盖测试用例表

逻辑覆盖类型	语句	条件	判定	测试用例编号	预期输入			预期输出
					A	B	X	
判定覆盖	S_1	$T_1T_2T_3T_4$	$T_{D1}T_{D2}$	1	8	2	12	X=6
	S_2	$F_1T_2F_3F_4$	$F_{D1}F_{D2}$	2	4	2	8	X=10

说明：满足判定覆盖的测试用例组，必定满足语句覆盖。虽然本例中判定覆盖率为100%，但依然存在测试的盲区，个别条件的真与假值没有取到。

(3) 条件覆盖

条件覆盖指的是设计若干测试用例来执行不同的条件结果，每个判断的每个条件的可能取值至少执行一次。条件覆盖率是指执行测试用例能够覆盖到原子条件的百分比，即被执行的原子条件和总的原子条件的比值。100%的条件覆盖要求测试覆盖到每一个原子条件语句分别取值为真和假的情况。

满足条件覆盖要求的测试用例如表 4.3 所示。

表 4.3　条件覆盖测试用例表

逻辑覆盖类型	语句	条件	判定	测试用例编号	预期输入			预期输出
					A	B	X	
条件覆盖		$T_1F_2T_3T_4$	$F_{D1}T_{D2}$	1	8	1	12	X=12
	S_2	$F_1T_2F_3F_4$	$F_{D1}F_{D2}$	2	4	2	8	X=10

说明：需要注意的是，条件覆盖并不比判定覆盖的覆盖能力更强。满足条件覆盖的测试用例，不一定就满足判定覆盖，所以也不一定就满足语句覆盖。

(4) 判定条件覆盖

判定条件覆盖是设计足够多的测试用例，使得判定中的每个条件都取到各种可能的值，而且每个判定表达式也都取到各种可能的结果。判定条件覆盖率指的是设计若干测试用例来执行条件结果和判定结果占执行测试用例套件能够覆盖的条件结果和判定结果的百分比。

满足判定条件覆盖要求的测试用例如表 4.4 所示。

表 4.4　判定条件覆盖测试用例表

逻辑覆盖类型	语句	条件	判定	测试用例编号	预期输入			预期输出
					A	B	X	
判定条件覆盖	S_1	$T_1T_2T_3T_4$	$T_{D1}T_{D2}$	1	8	2	12	X=6
	S_2	$F_1F_2F_3F_4$	$F_{D1}F_{D2}$	2	4	1	8	X=10

说明：从表面上看，判定条件覆盖测试了各个判定中的所有条件的取值，但实际上，编译器在检查含有多个条件的逻辑表达式时，某些情况下的某些条件将会被其他条件所掩盖。如 A>5&&B==2 这个判定中，只要 A>5 条件不满足，整个判定的结果就为 false，与条件 B==2 的取值没有任何关系。因此，判定条件覆盖也不一定能够完全检查出逻辑表达式中的错误，但是 100%的判定条件覆盖意味着 100%的判定覆盖和 100%的条件覆盖。

(5) 条件组合覆盖

条件组合覆盖是指设计足够多的测试用例覆盖每条语句中的原子条件所有可能的取值结果组合（即每个判定中的所有可能的原子条件取值组合至少执行一次），条件组合覆盖率是测试套件覆盖每条语句内的所有原子条件取值结果组合的百分比。

满足条件组合覆盖要求的测试用例如表 4.5 所示。

表 4.5　条件组合覆盖测试用例表

逻辑覆盖类型	语句	路径	条件	判定	测试用例编号	预期输入			预期输出
						A	B	X	
条件组合覆盖	S_1	R_1	$T_1T_2T_3T_4$	$T_{D1}T_{D2}$	1	8	2	12	X=6
	S_2	R_4	$F_1F_2F_3F_4$	$F_{D1}F_{D2}$	2	4	1	8	X=10
		R_3	$T_1F_2T_3F_4$	$F_{D1}T_{D2}$	3	8	1	2	X=2
		R_3	$F_1T_2F_3T_4$	$F_{D1}T_{D2}$	4	4	2	11	X=11

说明：100%条件组合覆盖意味着 100%判定条件覆盖。上面这组测试用例覆盖了所有 8 种条件取值的组合，覆盖了所有判定的真假分支，但是却丢失了一条路径 R_2。

(6) 路径覆盖

路径覆盖是指设计足够多的测试用例用来执行不同的路径，覆盖程序中所有可能的路径。路径覆盖率是测试套件执行的路径占总的路径的百分比。

满足路径覆盖要求的测试用例如表 4.6 所示。

表 4.6　路径覆盖测试用例表

逻辑覆盖类型	语句	路径	条件	判定	测试用例编号	预期输入			预期输出
						A	B	X	
路径覆盖	S_1	R_1	$T_1T_2T_3T_4$	$T_{D1}T_{D2}$	1	8	2	12	X=6
	S_2	R_4	$F_1F_2F_3F_4$	$F_{D1}F_{D2}$	2	4	1	8	X=10
		R_3	$T_1F_2T_3F_4$	$F_{D1}T_{D2}$	3	8	1	2	X=2
	S_1,S_2	R_2	$T_1T_2F_3F_4$	$T_{D1}F_{D2}$	4	6	2	10	X=7

说明：100%的路径覆盖可以确保100%的语句覆盖和判定覆盖。虽然前面一组测试用例满足了路径覆盖，但并没有覆盖程序中所有的条件组合，即满足路径覆盖的测试用例并不一定满足条件组合覆盖。在实际测试中，即使对于路径数很有限的程序，已经做到路径覆盖，仍然不能保证被测试程序的正确性，还需要采用其他测试方法进行补充。

4.2.2　能力目标

了解白盒测试中逻辑覆盖的基本概念和分类；了解条件、判定、可执行语句、路径的符号表示方式；掌握逻辑覆盖中六种覆盖方式下，测试用例的设计；掌握六种覆盖方式之间的包含关系。

4.2.3　任务驱动

1. 六种逻辑覆盖的联系和区别是什么？

2. 某个程序中的流程图如图 4.3 所示，求程序运行后 X 的值。使用逻辑覆盖法中六种覆盖方式分别设计测试用例。

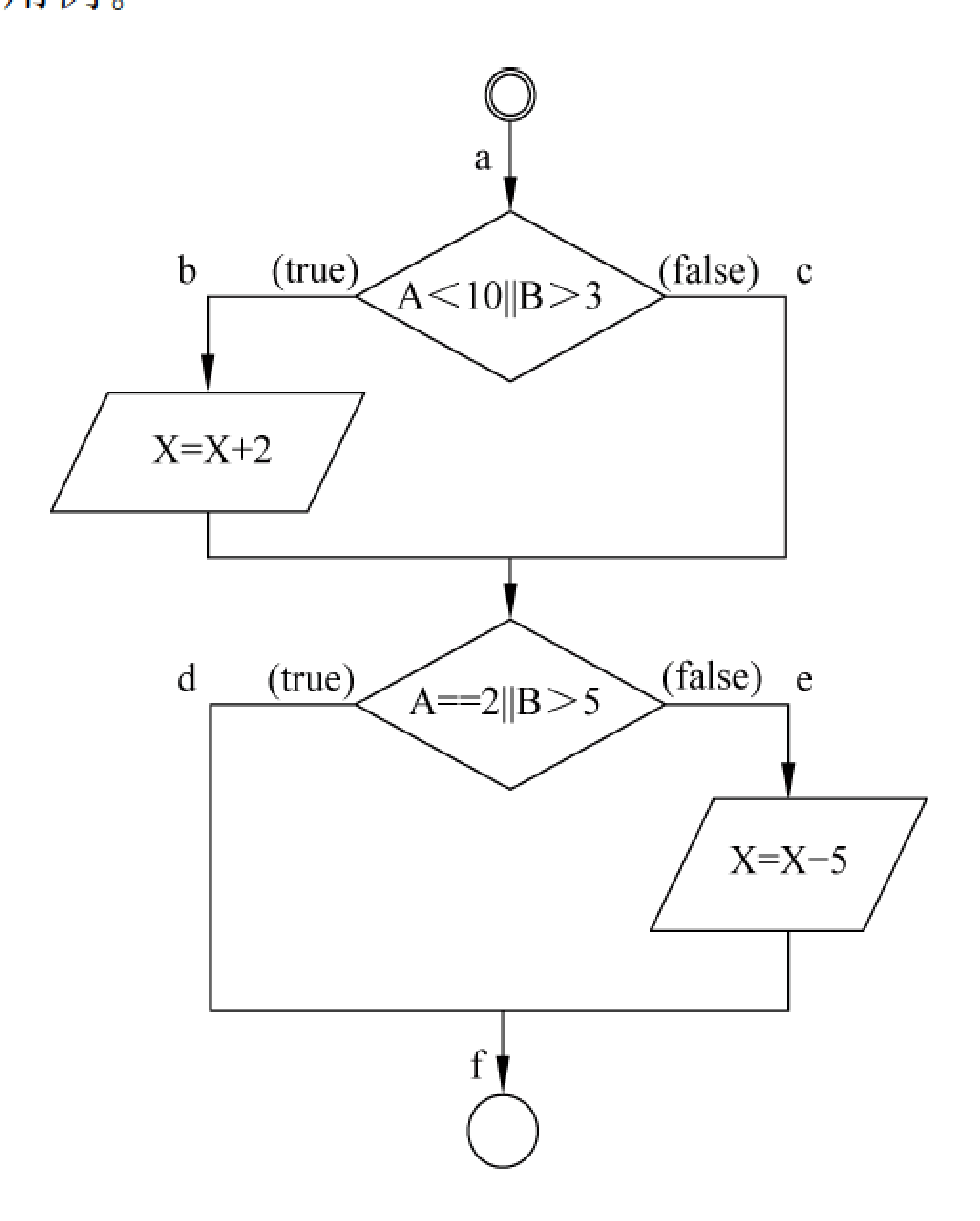


图 4.3　某程序流程图

4.2.4　实践环节

1. 对以下程序设计测试用例，画出程序流程图。实现语句覆盖、判定覆盖、条件覆盖、判定—条件覆盖、条件组合覆盖、路径覆盖，并给出 test 方法的返回值。

```
public int test(intx,inty,int z)
{
    intresult = 0;
    if((x>3)&&(y<10))
    {
        result = x * y;
    }
    if((z!= 4)||(y == 2))
    {
        result = z + y;
    }
    return result;
}
```

2. 针对以下 Java 语言编写的程序，画出程序流程图，对其进行判定—条件覆盖测试用例的设计。

```
public void Test( int m)
{
    int i,k;
    k  = Math.sqrt( m );
    for (i  =  2 ; i <=  k; i++)
    {
        if (m % i == 0)
        {
            break;
        }
        if ( i >=  k  +  1 )
        {
            System.out.println( m +  "is a selected number");
        }else{
            System.out.println( m + "is not a selected number");
        }
    }
}
```

4.3　独立路径测试法

4.3.1　核心知识

独立路径测试法是在程序控制流图的基础上，通过分析控制结构的环路复杂性，导出独立可执行路径的集合，从而设计测试用例的方法。设计出的测试用例要保证在测试中程序的每个可执行路径至少执行一次。

1. 获取独立路径的步骤

获取独立路径主要包括以下 4 个步骤。

(1) 画出程序的控制流图。控制流图是描述程序控制流的一种图示方法。

(2) 计算程序的环形复杂度。从程序的环形复杂度中可导出程序路径集合中的独立路径条数,这是确定程序中每个可执行路径至少执行一次所必需的测试用例数目的上界。

(3) 导出测试用例。根据环形复杂度和程序结构设计测试用例的数据输入和预期结果。

(4) 准备测试用例。确保独立路径集中的每一条路径的执行。

2. 控制流图

控制流图(简称流图)是对程序流程图进行简化后得到的,它可以更加突出地表示程序控制流的结构。控制流图中包括两种图形符号:节点和控制流线。节点由带标号的圆圈(○)表示;控制流线由带箭头的弧或线表示,可称为边,它与程序流程图中的流线一致,表明了控制的顺序,它代表程序中的控制流,通常标有名字。

程序的三大控制结构通常有顺序结构、分支结构和循环结构。其所对应的控制流图如图 4.4 所示。

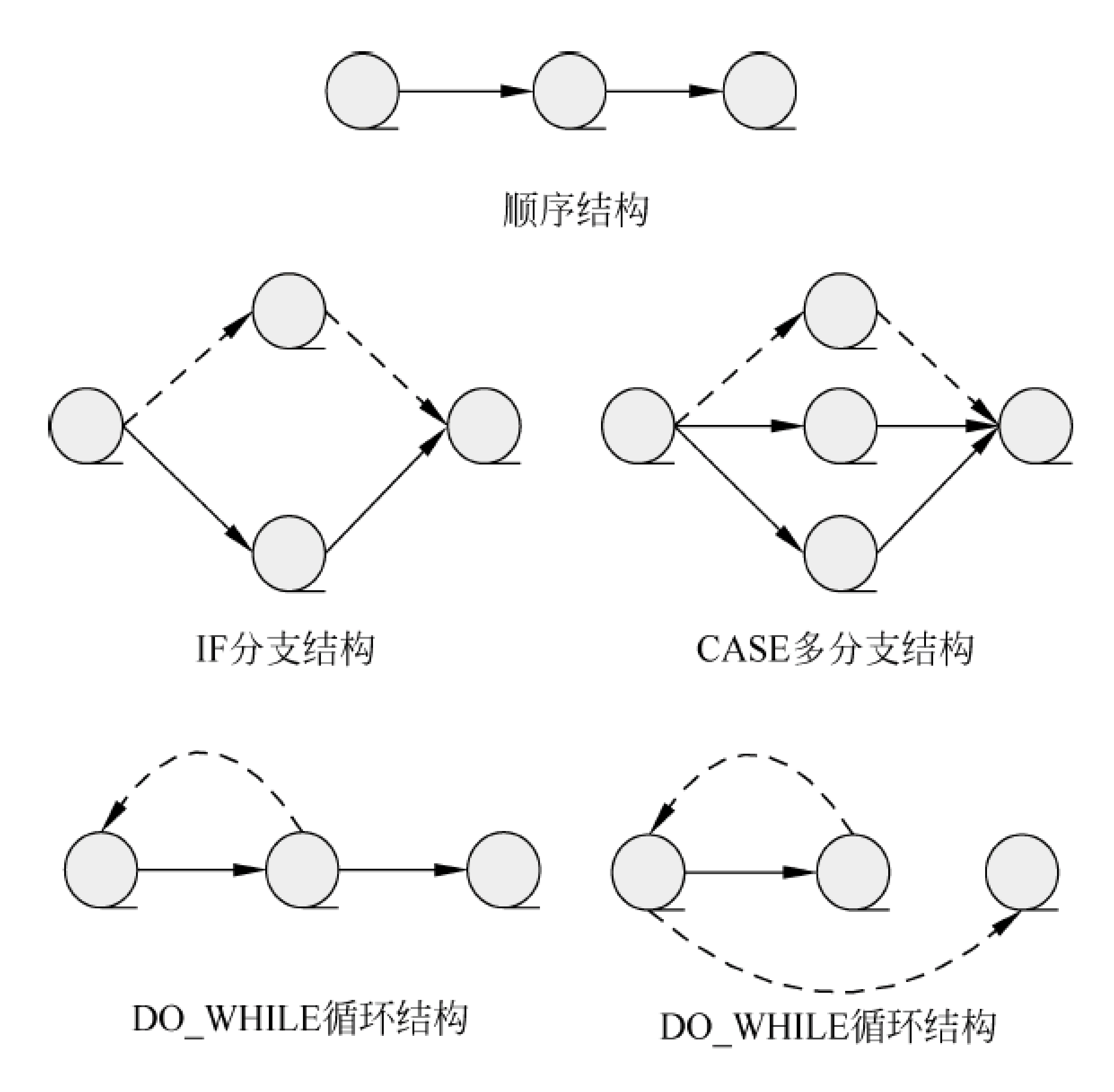


图 4.4　常见控制流图三大结构表示

3. 程序流程图向控制流图的转换规则

控制流图中的一些基本概念以及程序流程图向控制流图的转换规则如下。

(1) 包含条件的节点被称为判断节点,由判断节点发出的边必须终止于某一个节点。

(2) 用菱形框表示的判定条件内没有复合条件。如果判断中的条件表达式是由一个或多个逻辑运算符(OR,AND,…)连接的复合条件表达式,则需改为一系列只有单个条件的嵌套的判断。而一组顺序处理框可以映射为一个单一的节点。

(3) 控制流图中的箭头(边)表示了控制流的方向,类似于流程图中的流线,一条边必须终止于一个节点。

(4) 在选择或者多分支结构中分支的汇聚处，即使汇聚处没有执行语句也应该添加一个汇聚节点。

(5) 边和节点圈定的区域叫做区域，当对区域计数时，图形外的部分也应该记为一个区域。

以逻辑覆盖中的案例 1 为例，说明程序流程图向控制流图的转换规则。

案例 1 中的程序流程图如图 4.5 所示。

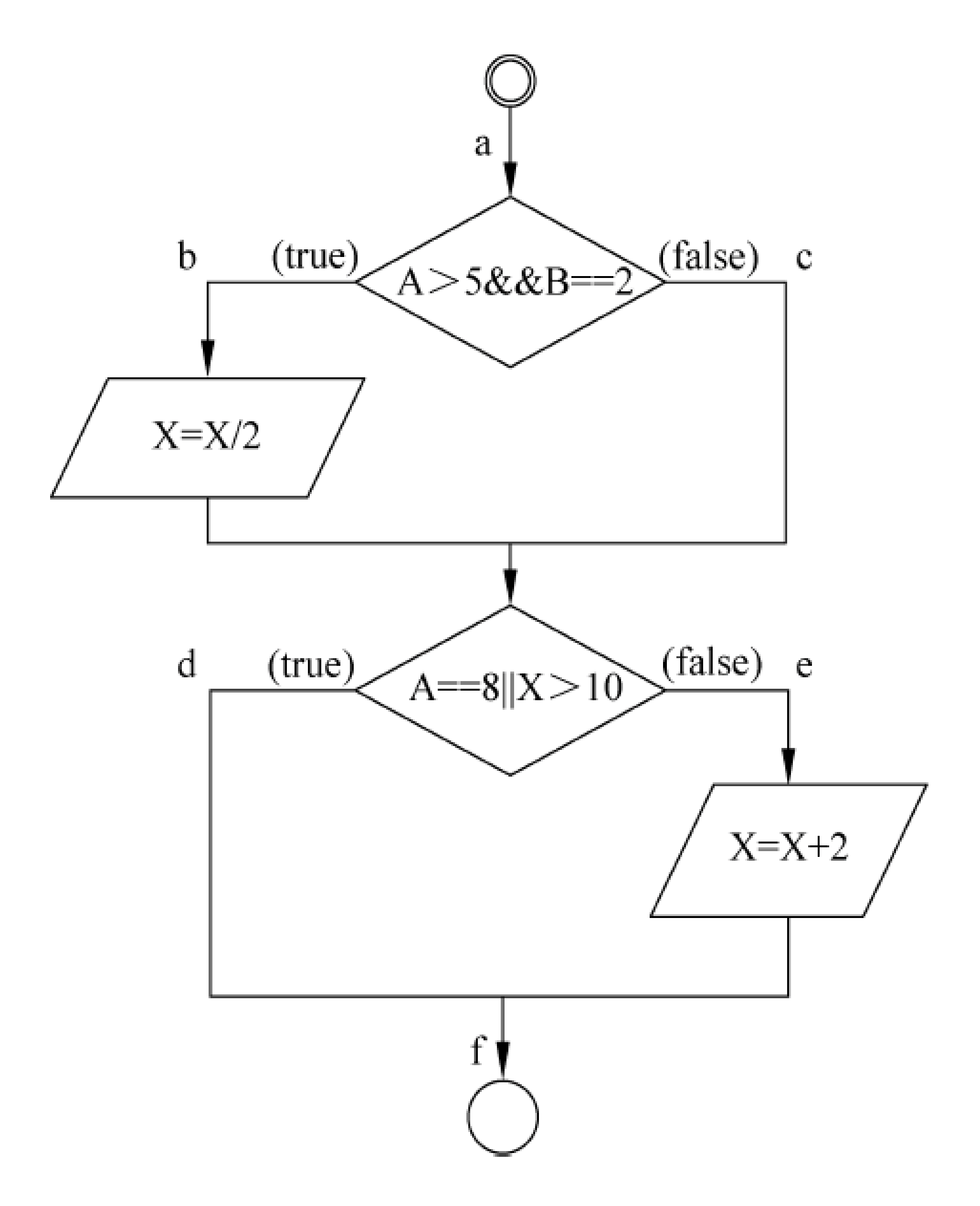


图 4.5　案例 1 中的程序流程图

由于该流程图中的两个判定都是由两个条件所组成，在转换为控制流图时，需要分解为单个条件，所以两个判定转换后的控制流图分别如图 4.6 和图 4.7 所示。

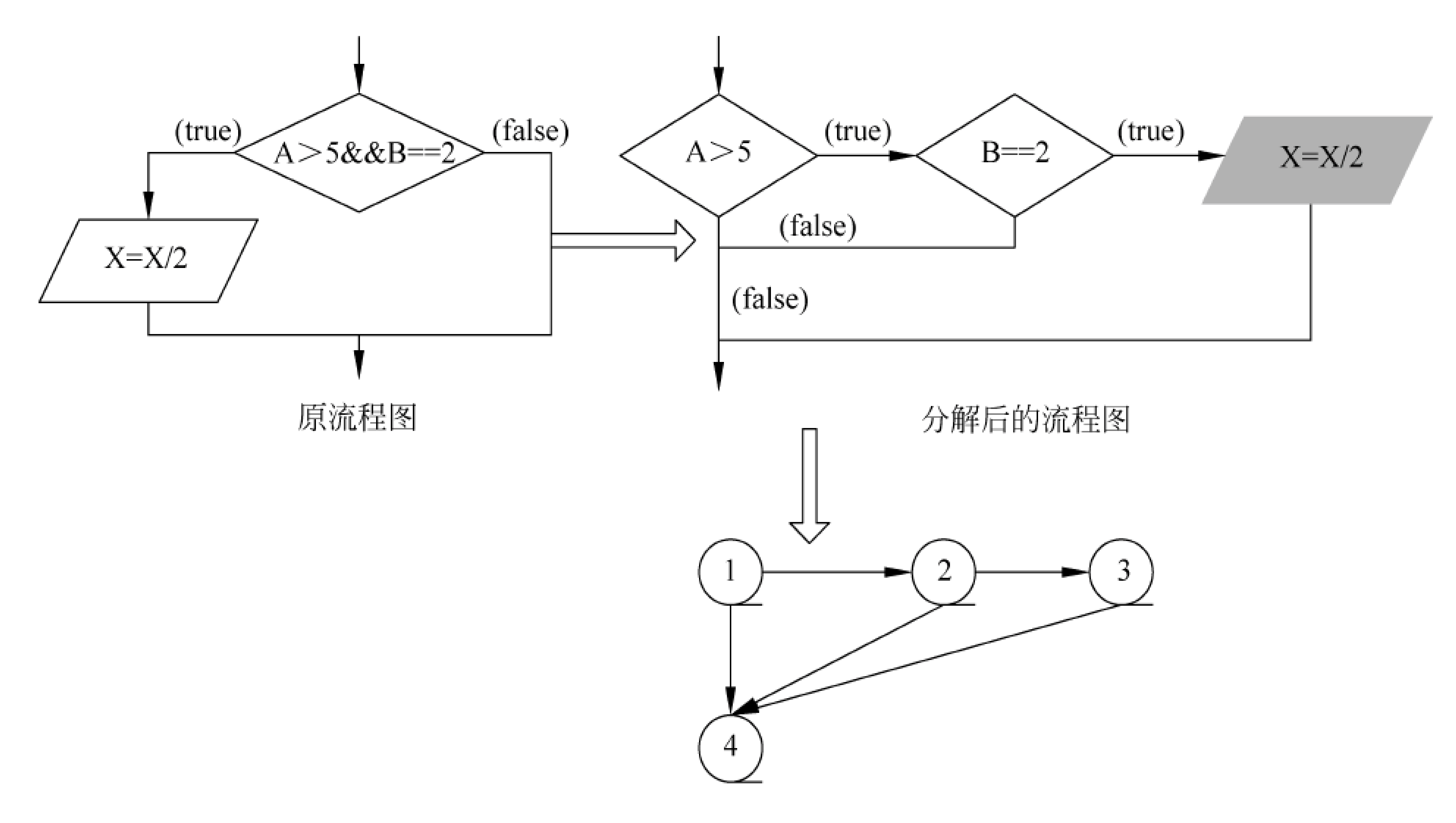


图 4.6　判定 1 转换成控制流图

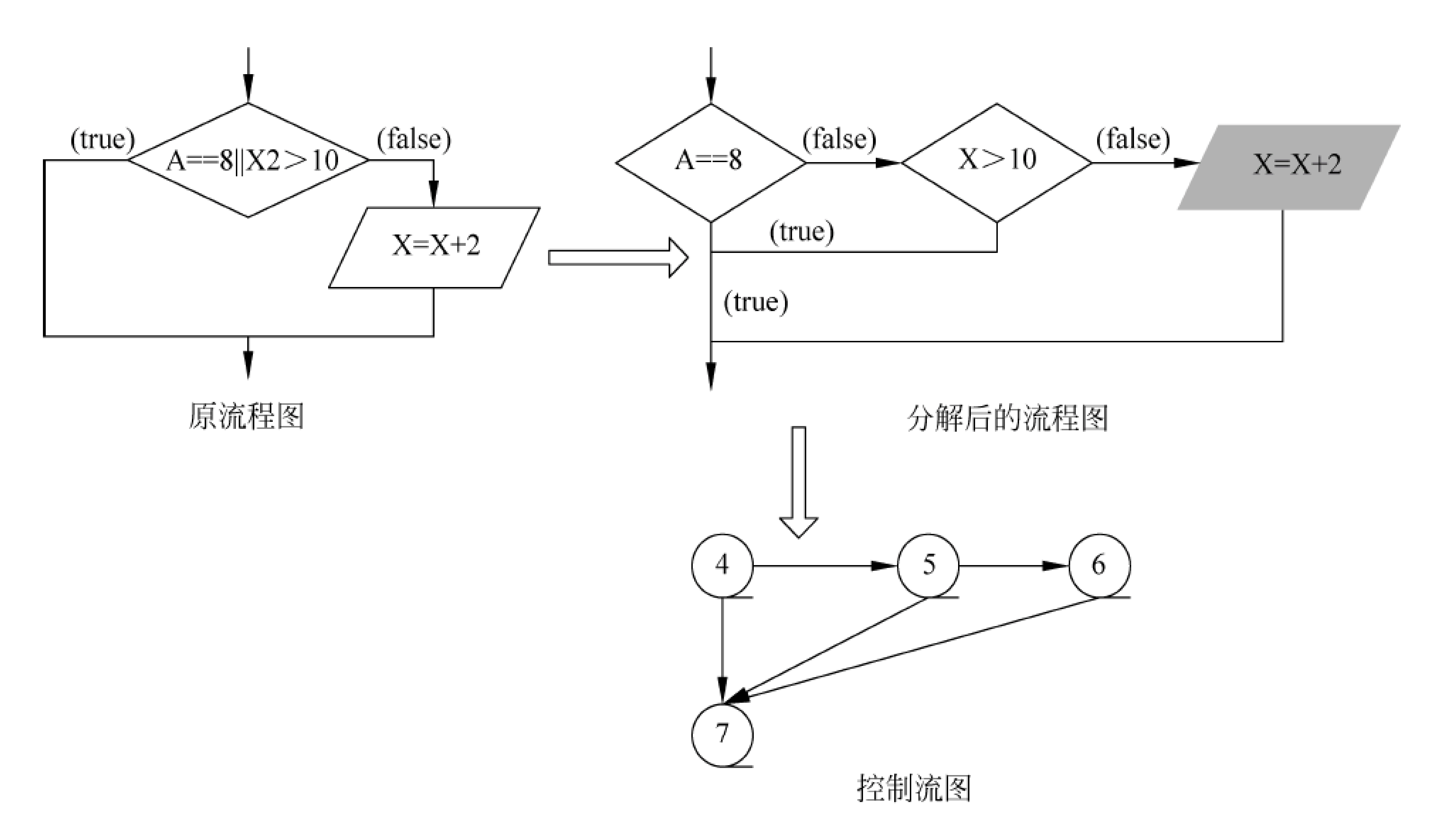


图 4.7　判定 2 转换成控制流图

可以得到整个程序流程图所对应的控制流图如图 4.8 所示。

4. 环形复杂度

环形复杂度也称为圈复杂度，它是一种为程序逻辑复杂度提供度量的标准。进行独立路径测试时，从程序的环形复杂度可导出程序独立路径集合中的独立路径条数。

独立路径是指包括一组以前没有处理的语句或条件的一条路径。这是确保程序中每个可执行路径至少执行一次所必需的测试用例数目的上界。

环形复杂度的三种计算方法：

(1) 控制流图中区域的数量等于程序的环形复杂度；

(2) 给定控制流图 G 的环形复杂度 V(G)，定义为 V(G)＝E－N＋2，E 是控制流图中边的数量，N 是控制流图中节点的数量；

(3) V(G)＝P＋1，P 是流图 G 中的判定节点数。

边和节点圈定的空间叫做区域，当对区域计数时，图形外的区域也应记为一个区域。由环形复杂度计算方法(1)：在案例 1 中得到的封闭区域有 4 个，图形外的区域 1 个，共 5 个区域，所以该案例中的环形复杂度 V(G)＝5，如图 4.9 所示。

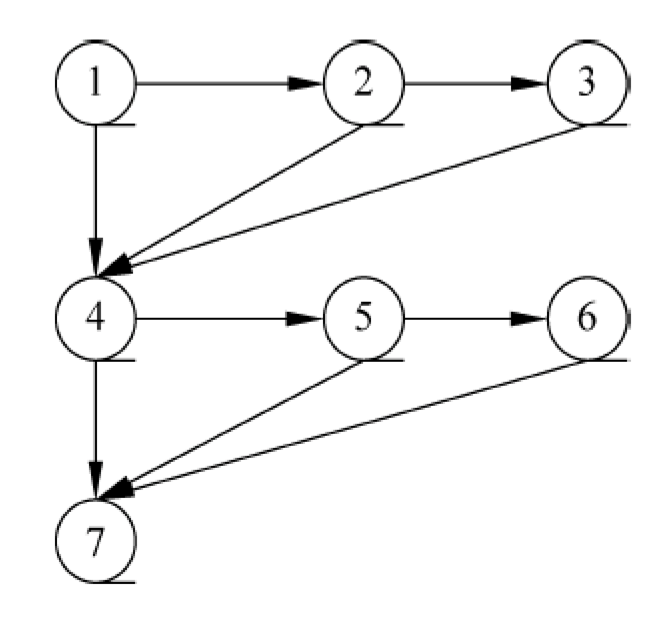


图 4.8　案例 1 控制流图

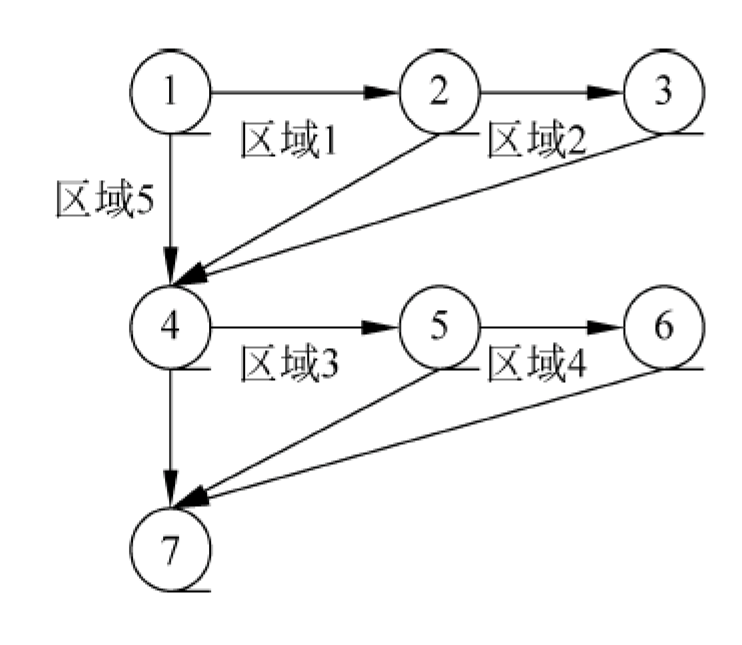


图 4.9　案例 1 控制流图中的区域

由环形复杂度计算方法(2)：该控制流图中有 10 条边，7 个节点，所以环形复杂度 V(G)=10－7＋2=5。

由环形复杂度计算方法(3)：该控制流图中判定节点有 4 个，节点代号分别为 1、2、4、5 号节点，所以环形复杂度 V(G)=4＋1=5。

本案例 1 中的独立路径个数为 5，而在程序流程图中得到的路径只有 4 个。5 条独立路径分别为：

① 1→4→7

② 1→2→4→7

③ 1→2→3→4→7

④ 1→2→4→5→7

⑤ 1→2→4→5→6→7

根据 5 条独立路径，设计测试用例如表 4.7 所示。

表 4.7　案例 1 独立路径测试法测试用例

白盒测试类型	测试用例编号	预期输入			覆盖路径	预期输出
		A	B	X		
独立路径测试	1	?	?	?	①	?
	2	8	1	2	②	X=2
	3	8	2	2	③	X=1
	4	7	1	2	④	X=2
	5	7	1	8	⑤	X=10

说明：虽然得到了 5 条独立路径，但是路径①中的条件在本案例中相互之间矛盾，故没有办法设计该条路径的测试用例，得到的测试用例个数为 4 条。

4.3.2　能力目标

了解白盒测试中独立路径测试的基本概念；掌握程序流程图到控制流图的转换规则；掌握环形复杂度的三种计算方式；针对独立路径设计测试用例。

4.3.3　任务驱动

1. 程序流程图和控制流图的区别是什么？如何转换？
2. 某个程序中的流程图如图 4.10 所示，求该程序的独立路径个数，并设计测试用例。

4.3.4　实践环节

针对 test 函数按照基本路径测试方法设计测试用例。

```
public inttest(inti_count, inti_flag)
{
    inti_temp = 0;
    while (i_count > 0)
    {
        if (0 == i_flag)
```

```
            {
                i_temp = i_count + 100;
                    break;
            }else{
                if (1 == i_flag)
                {
                    i_temp = i_temp + 10;
                }else{
                    i_temp = i_temp + 20;
                }
            }
            hi_count--;
        }
        return i_temp;
    }
```

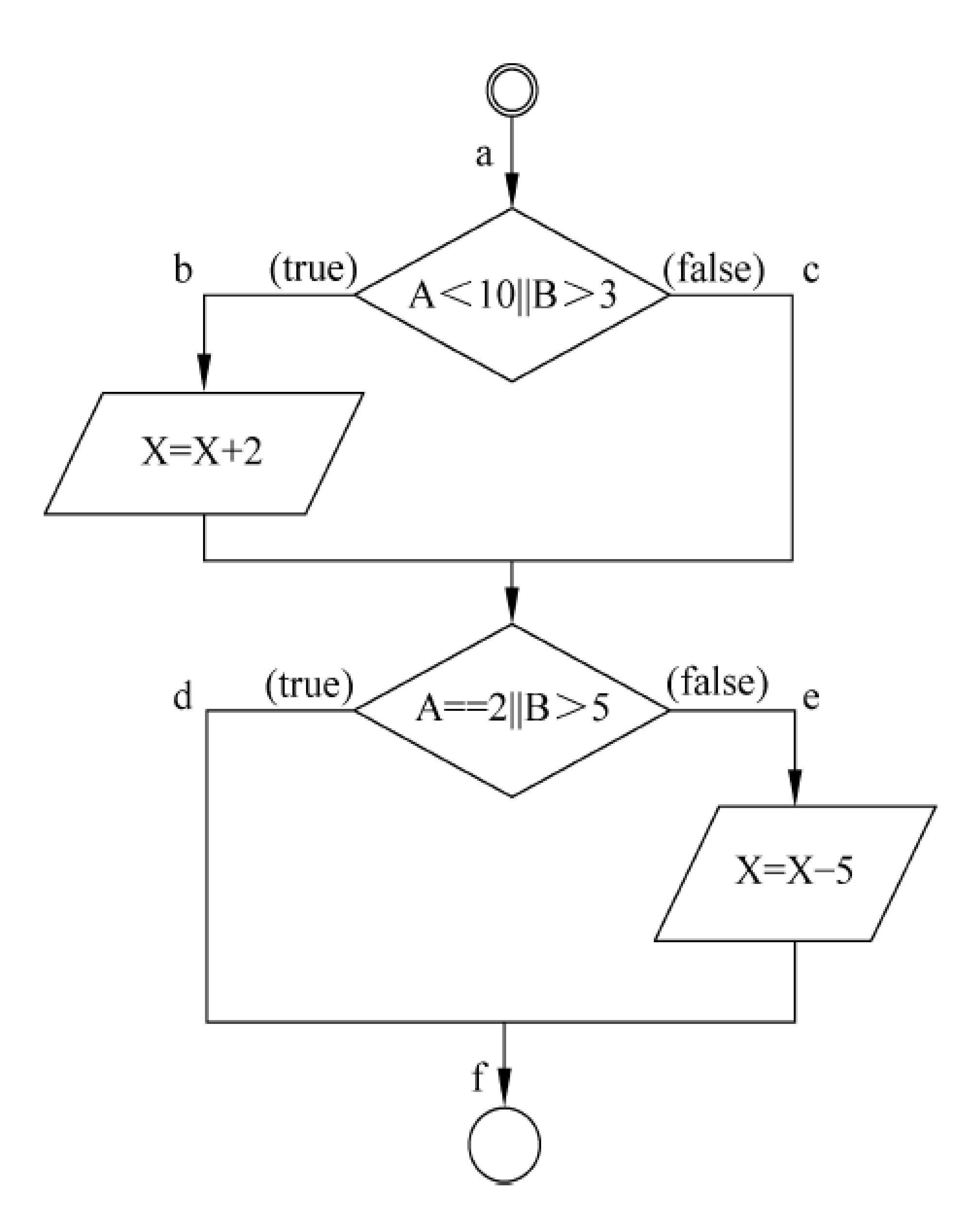


图 4.10　某个程序中的流程图

4.4　面向对象的白盒测试

4.4.1　核心知识

典型的面向对象程序具有继承、封装和多态的新特性，这使得传统的测试策略必须有所改变来适应程序语言的发展。封装是对数据的隐藏，外界只能通过被提供的操作来访问或修改数据，这样降低了数据被任意修改和读写的可能性，降低了传统程序中对数据非法操作的测试。继承是面向对象程序的重要特点，继承使得代码的重用率提高，同时也使错误传播的概率增加。继承使得传统测试遇见了这样一个难题：对继承的代码究竟应该怎样测试？多态使得面向对象程序对外呈现出强大的处理能力，但同时却使得程序内“同一”函数的行为复杂化，测试时不得不考虑不同类型具体执行的代码和产生的行为。

面向对象的程序是把功能的实现分布在类中。能正确实现功能的类，通过消息传递来协同实现设计要求的功能。正是这种面向对象程序风格，将出现的错误能精确地确定在某一具体的类上。

面向对象的白盒测试，把类作为一个单元来进行测试。测试分为两层：第一层考虑类中各独立方法的代码；第二层考虑方法之间的相互作用。

(1) 独立方法的白盒测试

面向对象的白盒测试，第一层是考虑各独立的方法，这可以与过程的测试采用同样的方法，两者之间最大的差别在于方法改变了它所在实例的状态，这就要取得隐藏的状态信息来估算测试的结果，传给其他对象的消息被忽略，而以桩来代替，并根据所传的消息返回相应的值，测试数据要求能完全覆盖类中代码，可以用传统的测试技术来获取。

(2) 方法之间的白盒测试

第二层要考虑一个方法调用本对象类中的其他方法和从一个类向其他类发送信息的情况。单独测试一个方法时，只考虑其本身执行的情况，而没有考虑动作的顺序问题，测试用例中加入了引发这些调用的信息，以检查它们是否正确地运行。对于同一类中方法之间的调用，一般只需要极少甚至不用附加数据，因为方法都是对类进行存取，故这一类测试的准则是要求遍历类的所有主要状态。

在类与类之间存在继承关系时，如果一个父类中有一个方法，子类继承了这个方法，尽管这个方法在父类中已经测试了这个方法，但是每个继承类也需要考虑对这个方法进行测试。一般从下面两个方面进行考虑。

① 继承的成员方法是否都不需要测试。一般来说，对父类中已经测试过的成员方法，两种情况需要在子类中重新测试，即继承的成员方法在子类中做了改动，或成员方法调用时改动过成员方法的部分内容。

② 对父类的测试是否能照搬到子类。多态有几种不同的形式，如参数多态、包含多态、过载多态。包含多态和过载多态在面向对象语言中通常体现在子类与父类的继承关系上。对具有包含多态的成员函数测试时，只需要在原有的测试分析的基础上扩大测试用例中输入数据的类型就可以解决。

面向对象软件的白盒测试，主要是针对软件设计中的类和对象来进行测试的。因此了解被测试的软件的类结构，是进行面向对象白盒测试的关键。由于封装的原则，在面向对象软件的类中的成员一般设计为私有或受保护类型，即类的属性和方法是无法从外部直接访问的，必须通过类中的公有方法来实现。因此设计测试用例时必须注意对这些公有成员的测试。为了实现对类中所有方法的有效测试，必须设计足够多的测试用例。

4.4.2　能力目标

了解面向对象的白盒测试的基本概念；掌握面向对象的白盒测试的测试重点。

4.4.3　任务驱动

1. 面向对象的程序设计有哪些特点？
2. 面向对象的白盒测试是否需要考虑各个对象的状态改变？

4.4.4 实践环节

1. 通过查阅资料,了解继承关系、接口关系、私有方法如何测试。
2. 了解软件行业内主流面向对象白盒测试工具。

4.5 小　　结

- 白盒测试使测试者能够看到被测源程序,可以分析被测程序的内部结构,按照程序内部逻辑测试程序,检查程序中每条通路是否按预定要求正确工作。
- 常见的白盒测试技术包括:逻辑覆盖法、独立路径测试法、面向对象的白盒测试等。
- 逻辑覆盖法可以分为:语句覆盖、判定覆盖、条件覆盖、判定条件覆盖、条件组合覆盖、路径覆盖。
- 独立路径测试法是在程序控制流图的基础上,通过分析控制构造的环路复杂性,导出基本可执行路径集合,从而设计测试用例的方法。设计出的测试用例要保证在测试中程序的每个可执行路径至少执行一次。
- 环形复杂度也称为圈复杂度,它是一种为程序逻辑复杂度提供定量尺度的软件度量。环形复杂度的三种计算方法:V(G)=区域的数量、V(G)=E−N+2、V(G)=P+1。
- 面向对象的白盒测试把类作为一个单元来进行测试。测试分为两层:第一层考虑类中各独立方法的代码;第二层考虑方法之间的相互作用。

习　题　4

1. 简述白盒测试用例的设计方法,并进行分析总结。

2. 分析归纳逻辑覆盖的各种策略,并比较每种覆盖的特点,分析在怎样的情况下采用何种覆盖方式。

3. 请针对以下代码按照各种覆盖方法设计测试用例。

```
if (a>5&& b==3 && (c>2 || d<7))
{
    Statement1;
}else{
    Statement2;
}
```

4. 对以下程序设计测试用例,画出程序流程图。使用独立路径测试法,进行测试用例的设计,并给出 getMax 方法的返回值。

```
public intgetMax(inta,intb,intc)
{
  intmax=a;
  if((a>b)&&(a>c))
```

```
  {
    max = a;
  }else{
    max = b;
    if(c > b){
       max = c;
    }
  }
  return max;
}
```

5. 面向对象的白盒测试和传统的白盒测试有什么区别？

6. 不属于白盒测试技术的是（　　）。
   A. 路径覆盖　　B. 判定覆盖
   C. 循环覆盖　　D. 边界值分析

7. 使用白盒测试方法时，确定测试数据应根据（　　）和指定的覆盖标准。
   A. 程序内部逻辑　　B. 程序的复杂度
   C. 使用说明书　　D. 程序的功能

8. 以程序内部的逻辑结构为基础的测试用例设计技术属于（　　）。
   A. 灰盒测试　　B. 数据测试
   C. 黑盒测试　　D. 白盒测试

9. 属于白盒测试技术的是（　　）。
   A. 决策表法　　B. 错误推断法
   C. 循环覆盖　　D. 边界值分析

10. 下列几种逻辑覆盖标准中，查错能力最强的是（　　）。
   A. 语句覆盖　　B. 判定覆盖
   C. 条件覆盖　　D. 条件组合覆盖

11. 发现错误能力最弱的是（　　）。
   A. 语句覆盖　　B. 判定覆盖
   C. 条件覆盖　　D. 路径覆盖

12. 在软件测试中，逻辑覆盖标准主要用于（　　）。
   A. 黑盒测试方法　　B. 白盒测试方法
   C. 灰盒测试方法　　D. 软件验收方法

13. 以下不属于逻辑覆盖的是（　　）。
   A. 语句覆盖　　B. 判定覆盖
   C. 条件覆盖　　D. 独立路径

14. 选择足够多的测试数据，使得判定表达式中的每个条件都取得各种可能的值，而且每个判定表达式也都取到各种可能的结果。满足这种测试条件的覆盖是（　　）。
   A. 判定覆盖　　B. 条件覆盖
   C. 判定条件覆盖　　D. 条件组合覆盖

15. 针对布尔表达式 A&&(B||C) 执行逻辑覆盖测试,测试用例至少需要(　　)种组合才能满足条件组合覆盖的要求。

A. 6　　B. 4

C. 8　　D. 12

16. 针对下面的语句段,采用语句覆盖法完成测试用例设计,测试用例见下表,对表中的空缺项(TRUE 或者 FALSE),正确的选择是(　　)。

语句段:

```
if (A && (B||C)) x = 1;
      else x = 0;
```

用例表:

	用例 1	用例 2
A	TRUE	FALSE
B	①	FALSE
C	TRUE	②
A &&(B||C)	③	FALSE

A. ①TRUE ②FALSE ③TRUE　　B. ①TRUE ②FALSE ③FALSE

C. ①FALSE ②FALSE ③TRUE　　D. ①TRUE ②TRUE ③FALSE

17. 在面向对象的白盒测试中,测试的单元是(　　)。

A. 类　　B. 接口

C. 函数　　D. 属性

18. 面向对象的白盒测试中,第一层考虑测试的对象是(　　)。

A. 类　　B. 各独立的方法

C. 调用其他的方法　　D. 状态

第 5 章

软件测试管理及自动化测试

主要内容

- 软件测试管理计划
- 软件测试管理过程
- 软件测试管理的主要功能
- 软件测试管理周期
- 自动化测试的优点与局限
- 自动化测试的生命周期

在本章中主要掌握软件测试管理计划的作用；掌握软件测试管理过程、主要功能、内容；掌握软件缺陷管理的定义以及缺陷的属性；了解软件测试管理周期；了解自动化测试发展的必然性，自动化测试的优点与局限以及自动化测试的生命周期。

5.1 软件测试管理

5.1.1 核心知识

随着IT技术的迅速发展，计算机在各行各业日益广泛的应用，软件产品的不断推出，计算机软件已经越来越深入到人们的生活中，人们对计算机软件质量的要求也就越来越高。如果软件存在故障，将可能造成人力、物力和财力的巨大浪费；如果软件的质量不高，其维护费用不仅将大大超过其开发费用，而且会使维护变得很困难，甚至将可能造成不可弥补的损失。由此可见，人们为了保证软件产品的质量，必须对计算机软件进行测试。由于软件测试至今仍令人捉摸不定，为确保测试工作的顺利进行，就要对其进行有效的管理。软件测试管理是一种活动，可以对各阶段的测试计划、测试案例、测试流程进行管理、跟踪，记录其结果，并将结果反馈给系统的开发者和管理者。同时将测试人员发现的错误立刻记录下来，生成问题报告并对之进行管理。所以采用软件测试管理方法，可以为软件企业提供一个多阶段、逐步递进的实施方案。通过测试管理方法，软件企业可以用有限的时间和成本完成软件开发，确保软件产品的质量，进一步提高计算机软件在市场上的竞争能力。因此，近年来软件测试管理愈来愈受到IT行业的关注。经过多年努力，软件测试管理正在走上一条正规之路。

实践证明，对软件进行测试管理可以及早地发现错误，避免大规模返工，降低软件开发

费用。为确保最终件质量符合要求,必须进行测试与管理。对于不同企业的不同类产品、同一企业的不同类产品,或不同企业的同一类产品,其各阶段结果的形式与内容都会有很大的不同。所以对于软件测试管理,除了要考虑测试管理开始的时间、测试管理的执行者、测试管理技术如何有助于防止错误的发生、测试管理活动如何被集成到软件过程的模型中之外,还必须在测试之前,制订详细的测试管理计划,充分实现软件测试管理的主要功能,缩短测试管理的周期。

1. 软件测试管理计划

一个成功的测试开始于一个全面的测试管理计划。因此,在每次测试之前应做好详细的测试管理计划:首先应该了解被测对象的基本信息,选择测试的标准级别,明确测试管理计划标识和测试管理项。在定义了被测对象的测试管理目标、范围后必须确定测试管理所使用的方法,即提供技术性的测试管理策略和测试管理过程。在测试管理计划中,管理者应该全面了解被测试对象的系统方法、语言特征、结构特点、操作方法和特殊需求等,以便确定必要的测试环境,包括测试硬件、软件及测试环境的建立等。而且,在测试管理计划中还应该制订一份详细的进度计划,如:测试管理的开始段、中间段、结束段及测试管理过程每个部分的负责人等。由于任何一个软件不可能没有缺陷、系统运行时不出现故障,所以在测试管理计划中还必须考虑到一些意外情况,也就是说,当问题发生时应该如何处理。因为测试管理具有一定难度,所以对测试管理者应进行必要的测试设计、工具、环境等的培训。最后,还必须确定认可和审议测试管理计划的负责人员。

2. 软件测试管理过程

制订完软件测试管理计划后,就可以根据计划执行测试管理。软件测试管理的过程如图 5.1 所示。

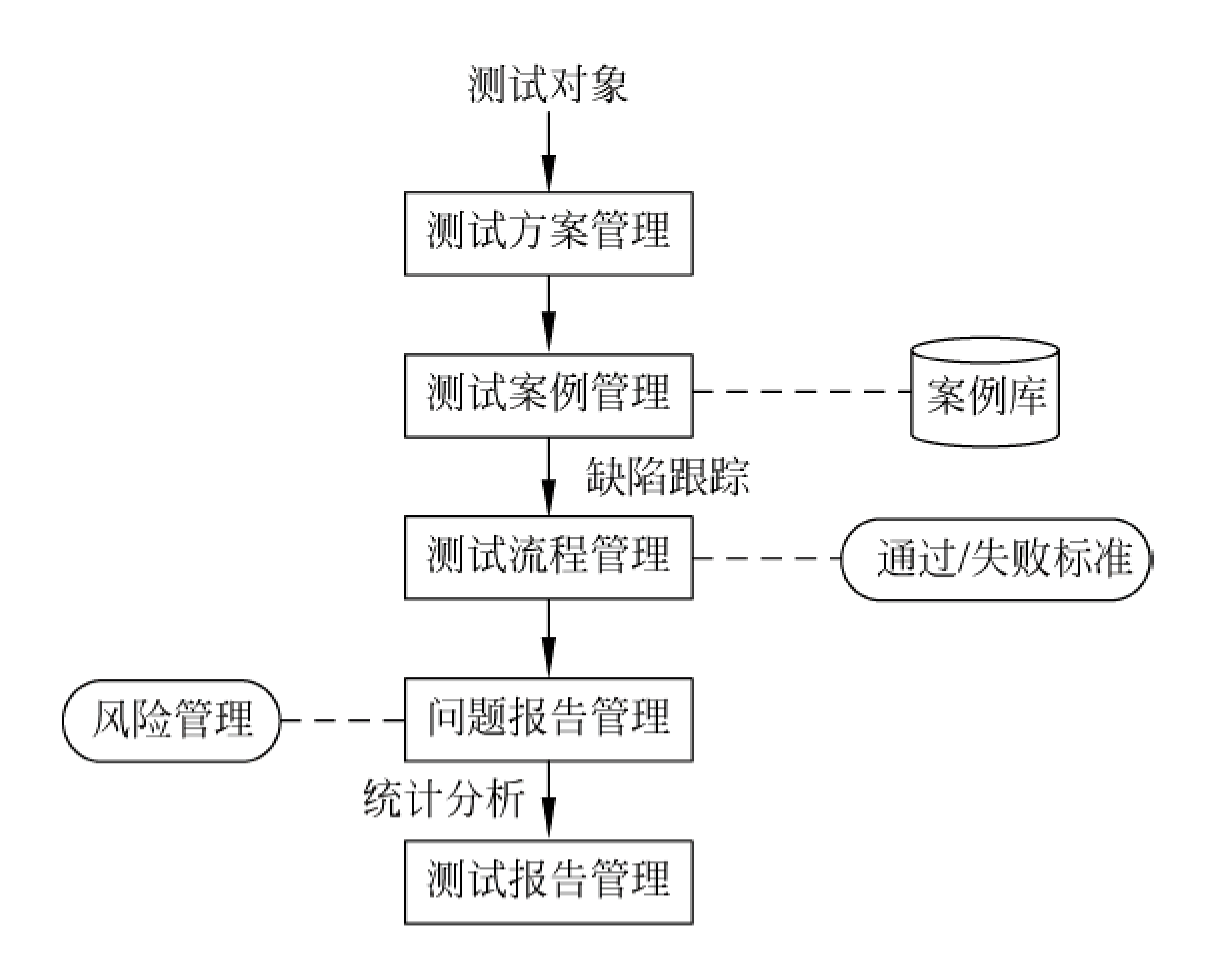


图 5.1 **软件测试管理过程图**

软件测试管理过程对测试过程中每个状态进行记录、跟踪和管理,并提供相关的分析和统计功能,生成和打印各种分析统计报表。通过对详细记录的分析,形成较为完整的软件测试管理文档,保障软件在开发过程中避免同样的错误再次发生,从而提高软件开发质量。

3. 软件测试管理内容

具体的测试管理内容如下。

(1) 测试方案管理。包括单元测试、集成测试和产品测试的测试计划的录入、修改、删除、查询和打印。

(2) 测试案例管理。包括测试案例的增、删、改、拷贝和查询；测试案例测试情况的管理，如测试状态包括未测试、测试中、已测试；测试结果分为通过、未实现、存在问题等；测试案例输入、编号和归档。

(3) 测试流程管理。包括测试进度管理、测试流程标识、测试日志及状态报告。

(4) 问题报告管理。包括问题报告处理流程(问题报告到整改报告)、实现问题报告与测试案例的关联。

(5) 测试报告管理。生成单元测试、集成测试和产品测试的测试报告。

除了以上这些，在测试管理过程中还应对人员和环境资源进行管理。

(6) 软件测试管理人员。为了实现软件测试管理，需要组成一个专门的测试管理队伍，队伍中的人员都能够胜任他们所担任的角色。另外，还需确保每种角色的人员应该具有必要的权利以完成他们的任务。同时为了能够获得较高的效率，每个测试管理参与者应该最大限度地发挥出其技术能力。

(7) 环境资源。包括硬件资源和软件资源，它们是提供测试管理的基础。每类资源都可以用四个特征来说明，包括资源描述、可用性说明、需要该资源的时间及该资源被持续使用的时间。

4. 软件缺陷管理

软件缺陷跟踪管理是测试工作的一个重要部分，测试的目的是为了尽早发现软件系统中的缺陷，而对软件缺陷进行跟踪管理的目的是确保每个被发现的缺陷都能够及时得到处理。软件测试过程简单说就是围绕缺陷进行的、对缺陷的跟踪管理。

软件缺陷属性包括缺陷标识、缺陷类型、缺陷严重程度、缺陷产生可能性、缺陷优先级、缺陷状态、缺陷起源、缺陷来源、缺陷原因。

(1) 缺陷标识。标记某个缺陷的唯一的表示，可以使用数字序号表示。

(2) 缺陷类型。根据缺陷的自然属性划分缺陷种类。缺陷类型如表 5.1 所示。

表 5.1　缺陷类型

缺陷类型	描　　述
功能	影响了各种系统功能、逻辑的缺陷
用户界面	影响了用户界面、人机交互特性，包括屏幕格式、用户输入灵活性、结果输出格式等方面的缺陷
文档	影响发布和维护，包括注释、用户手册、设计文档
软件包	由于软件配置库、变更管理或版本控制引起的错误
性能	不满足系统可测量的属性值，如执行时间、事务处理速率等
系统(模块)接口	与其他组件、模块或设备驱动程序、调用参数、控制块或参数列表等不匹配、冲突

(3) 缺陷严重程度。指因缺陷引起的故障对软件产品的影响程度，所谓"严重性"指的是在测试条件下，一个错误在系统中的绝对影响。如软件缺陷严重等级如表 5.2 所示。

表 5.2 软件缺陷严重等级

缺陷严重等级	描　　述
致命	系统任何一个主要功能完全丧失、用户数据受到破坏、系统崩溃、悬挂、死机，或者危及人身安全
严重	系统的主要功能部分丧失、数据不能保存，系统所提供的功能或服务受到明显的影响
一般	系统的部分功能没有完全实现，但不影响用户的正常使用，例如：提示信息不太准确，或用户界面差、操作时间长等一些问题
较小	使操作者不方便或遇到麻烦，但它不影响功能的操作和执行，如个别的不影响产品理解的错别字、文字排列不整齐等一些小问题

(4) 缺陷产生的可能性。指缺陷在产品中发生的可能性，通常可以用频率来表示，如表 5.3 所示。

表 5.3 软件缺陷可能性

缺陷产生的可能性	描　　述
总是	总是产生这个软件缺陷，其产生的频率是 100%
通常	按照测试用例，通常情况下会产生这个软件缺陷，其产生的频率大概是 80%～90%
有时	按照测试用例，有的时候产生这个软件缺陷，其产生的频率大概是 30%～50%
很少	按照测试用例，很少产生这个软件缺陷，其产生的频率大概是 1%～5%

(5) 缺陷优先级。指缺陷必须被修复的紧急程度。“优先级”的衡量抓住了在严重性中没有考虑的重要程度因素。缺陷优先级如表 5.4 所示。

表 5.4 缺陷优先级

缺陷优先级	描　　述
立即解决(P1 级)	缺陷导致系统几乎不能使用或测试不能继续，需立即修复
高优先级(P2 级)	缺陷严重，影响测试，需要优先考虑
正常排队(P3 级)	缺陷需要正常排队等待修复
低优先级(P4 级)	缺陷可以在开发人员有时间的时候被纠正

(6) 缺陷状态。指缺陷通过一个跟踪修复过程的进展情况，也就是在软件生命周期中的状态基本定义，如软件缺陷状态如表 5.5 所示。

表 5.5 软件缺陷状态

缺 陷 状 态	描　　述
激活或打开(Active or Open)	问题还没有解决，存在源代码中，确认“提交的缺陷”，等待处理，如新报的缺陷
已修正或修复(Fixed or Resolved)	已被开发人员检查、修复过的缺陷，通过单元测试，认为已解决但还没有被测试人员验证
关闭或非激活(Closed or Inactive)	测试人员验证后，确认缺陷不存在之后的状态
重新打开(Reopen)	测试人员验证后，还依然存在的缺陷，等待开发人员进一步修复
推迟(Deferred)	这个软件缺陷可以在下一个版本中解决
保留(on Hold)	由于技术原因或第三者软件的缺陷，开发人员不能修复的缺陷
不能重现(Cannot duplicate)	开发不能复现这个软件缺陷，需要测试人员检查缺陷复现的步骤

续表

缺 陷 状 态	描　　述
需要更多信息(Need more infor)	开发能复现这个软件缺陷,但开发人员需要一些信息,例如:缺陷的日志文件、图片等
重复(Duplicate)	这个软件缺陷已经被其他的软件测试人员发现
不是缺陷(Not a bug)	这个问题不是软件缺陷
需要修改软件规格说明书(Spec modified)	由于软件规格说明书对软件设计的要求,软件开发人员无法修复这个软件缺陷,必须修改软件规格说明书

(7) 缺陷起源。缺陷引起的故障或事件第一次被检测到的阶段,如软件缺陷起源如表 5.6 所示。

表 5.6　软件缺陷起源

缺陷起源	描　　述
需求	在需求阶段发现的缺陷
构架	在系统构架设计阶段发现的缺陷
设计	在程序设计阶段发现的缺陷
编码	在编码阶段发现的缺陷
测试	在测试阶段发现的缺陷
用户	在用户使用阶段发现的缺陷

(8) 缺陷来源。指缺陷所在的地方,如文档、代码等,如软件缺陷来源如表 5.7 所示。

表 5.7　软件缺陷来源

缺陷来源	描　　述
需求说明书	需求说明书的错误或不清楚引起的问题
设计文档	设计文档描述不准确和需求说明书不一致的问题
系统集成接口	系统各模块参数不匹配、开发组之间缺乏协调引起的缺陷
数据流(库)	由于数据字典、数据库中的错误引起的缺陷
程序代码	纯粹在编码中的问题所引起的缺陷

(9) 缺陷根源。指造成上述错误的根本因素,以寻求软件开发流程的改进、管理水平的提高,如软件缺陷根源如表 5.8 所示。

表 5.8　软件缺陷根源

缺陷根源	描　　述
测试策略	错误的测试范围,误解了测试目标,超越测试能力等
过程、工具和方法	无效的需求收集过程,过时的风险管理过程,不适用的项目管理方法,没有估算规程,无效的变更控制过程等
团队/人	项目团队职责交叉,缺乏培训,没有经验的项目团队,缺乏士气和动机不纯等
缺乏组织和通信	缺乏用户参与,职责不明确,管理失败等
硬件	硬件配置不对、缺乏,或处理器缺陷导致算术精度丢失,内存溢出等
软件	软件设置不对、缺乏,或操作系统错误导致无法释放资源,工具软件的错误,编译器的错误,2000"千年虫"问题等
工作环境	组织机构调整,预算改变,工作环境恶劣,如噪声过大

软件缺陷的生命周期如图 5.2 所示。

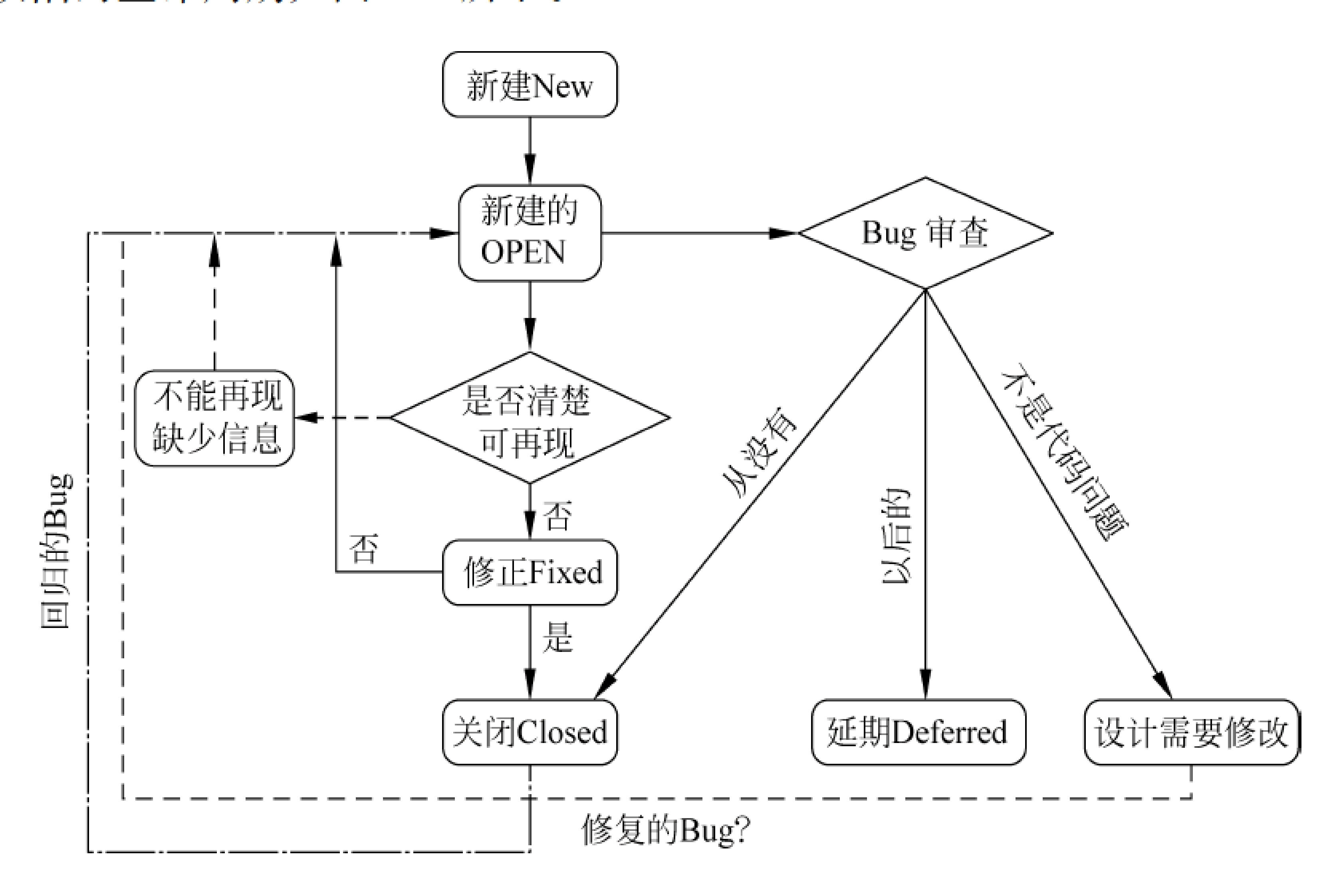


图 5.2　软件缺陷的生命周期

5. 软件测试管理的主要功能

测试管理的主要功能包括以下三方面。

(1) 测试控制对象的编辑和管理。测试控制对象包括测试方案、测试案例、各案例的具体测试步骤、问题报告、测试结果报告等,该部分主要是为各测试阶段的控制对象提供一个完善的编辑和管理环境。

(2) 测试流程控制和管理。测试流程的控制和管理是基于科学的流程和具体的规范来实现的,并利用该流程和规范严格约束和控制整个产品的测试周期,以确保产品的质量。整个过程避免了测试人员和开发设计人员之间面对面的交流,减少了以往测试和开发之间难免的摩擦和矛盾,提高了工作效率。

(3) 统计分析和决策支持。在系统建立的测试数据库的基础上,进行合理的统计分析和数据挖掘。例如,根据问题分布的模块、问题所属的性质、问题的解决情况等方面的统计分析使项目管理者全面了解产品开发的进度,产品开发的质量,产品开发中问题的聚集,为决策管理提供支持。例如,设计人员在遇到问题时可以到案例库中查找类似问题的解决办法等。

6. 软件测试管理周期

任何程序,无论大小都可能会有错误发生。每一个新版本都需要进行新特性的测试和其他特性的一些回归测试。所以软件测试管理具有周期性,如图 5.3 所示。

测试管理人员在接受一个测试管理任务后,除了要制订周密的测试管理计划,还要进行测试方案管理,并且对测试人员所做的测试活动予以记录,做好测试流程的管理。同时,对发现的缺陷予以标识,一方面反馈给提交测试的人员;另一方面将存在的问题和缺陷存入案例库,直至测试通过。

7. 软件测试管理的应用

软件测试是一个完整的体系,主要由测试规划、测试设计、测试实施、资源管理等相互关

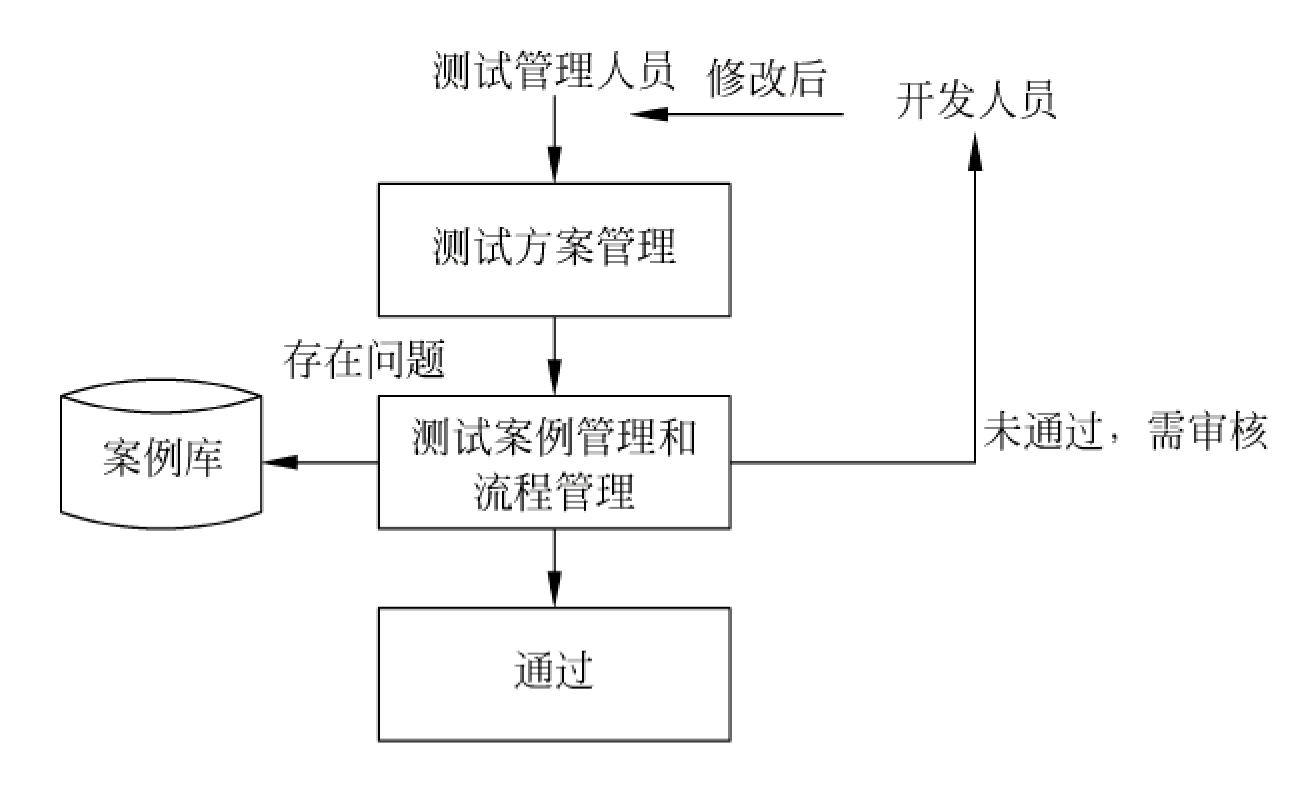


图 5.3　软件测试管理周期

联、相互作用的过程构成。软件测试管理系统可以对各过程进行全面控制。具体的实现过程如下。

（1）按照国际质量管理标准，建立适合本公司的软件测试管理体系，以提高公司开发的软件质量，并降低软件开发及维护成本。

（2）建立、监测和分析软件测试过程，以有效地控制、管理和改进软件测试过程，监测软件质量，从而确定交付或发布软件的时间。

（3）制订合理的软件测试管理计划，设计有效的测试案例集，以尽可能发现软件缺陷，并组织、管理和应用庞大的测试案例集。

（4）在软件测试管理过程中，管理者、程序员、测试员（含有关客户人员）协同工作，及时解决发现软件问题。

（5）对于软件测试中发现的大量软件缺陷，进行合理的分类以分清轻重缓急。同时进行原因分析，并做好相应的记录、跟踪和管理工作。

（6）建立一套完整的文档资料管理体系。因为，软件测试管理很大限度上是通过对文档资料的管理来实现的。软件测试每个阶段的文档资料是以后阶段的基础，又是对前面阶段的复审。

8. 软件测试管理工具简介

在软件开发生命周期中软件测试管理工具是非常重要的手段。为了便于对制定的测试方案、编写测试案例和测试步骤等各个阶段进行有效的控制和管理；提高软件开发和产品测试的管理水平，保证软件产品质量；大幅度降低测试人员的工作量和重复劳动，提高测试人员的工作效率和积极性，在此，介绍一些比较流行的软件测试管理工具。

（1）TMS 软件测试管理系统

TMS 测试管理系统管理功能全面，对测试流程的设计科学、规范、合理。系统的开发是在充分借鉴了 Microsoft、Nortel 等国际知名大公司在测试领域尤其是测试流程管理方面的经验，参考了 SQA Manager 等国外知名测试管理软件，并结合开发人员在业界的经验和对国内软件开发现状的把握等基础上开发而成，非常贴近国内用户的需求，具有很大的测试案例、测试步骤的编辑和管理功能，问题（缺陷）的跟踪处理功能，所有输出结果自动生成 Word 文档的功能，同时有强大的统计分析、决策支持能力。该系统技术实现上采用 Web/

Browser 开发模式,使用维护方便,具有良好的性价比。

(2) Test Management Workshop 测试管理工具

该系统定义了一个良好的表单归档机制,并且支持这些表单的交叉引用。通过关键项目文档中内建的关联设计,用户可以根据不同的线索追踪和调用相关的文档。同时,所有文档均被置于严格的安全控制之下,而且客户端支持浏览器方式操作。

(3) Jactus Labs 测试管理工具

Jactus Labs 的测试管理工具为了适应数以百计的用户,有一个中心数据储存库,所有的用户可以共享并存取主要的信息——测试脚本、缺陷及报告书。该测试管理工具把测试计划、测试执行和缺陷跟踪三者有机地结合在一起,同时为了更多的灵活性还采用了开放式测试结构(Open Test Architecture,OTA)。它利用 Microsoft Access 数据库缩小安装,并利用符合行业标准的关系数据库包括 Oracle、Microsoft SQL Server 和 Sybase 来扩大安装测试管理工具。对于每一件测试案例,它都会列出用户操作的顺序、案例描述、状态和预期的结果,这些信息都可以逐步填在一张校验表里并被记录在所有测试案例文件中,从而使测试过程更合理、统一。

(4) i-Test 软件测试管理系统

i-Test 系统采用 B/S 结构,可以安装在 Web 服务器上,项目有关人员都可以在不同地点通过 Internet 同时登录和使用,协同完成软件测试,减少为了集中人员而出差所产生的费用。同时,该系统提供相应的自动化功能,可高效编写、查询和引用测试用例,快速填写、修改和查询软件缺陷报告,并提供相关的分析和统计功能,生成和打印各种分析统计报表。

这些软件测试管理工具可以为企业商业系统提供全面的、综合的测试管理解决方案,并可以控制和管理所有的测试工作来确保测试是一个有组织的、规范文档化的和全面的测试活动。

5.1.2 能力目标

了解软件测试管理的作用;掌握软件测试管理过程和主要管理内容;掌握软件缺陷管理,软件缺陷的基本属性;掌握软件测试管理的主要功能;了解主流的软件测试管理工具。

5.1.3 任务驱动

1. 如何制订软件测试管理计划?
2. 软件测试管理的主要内容有哪些?
3. 软件测试管理的主要功能是什么?
4. 什么是软件缺陷?软件缺陷有哪些属性?

5.1.4 实践环节

1. 软件缺陷的等级如何划分?
2. 进入软件公司调研,并阐述软件缺陷管理的原则。
3. 选择一种测试管理工具,通过实际操作,了解其工作流程。

5.2　自动化测试

5.2.1　核心知识

自动化测试是软件测试发展的一个必然趋势。随着软件技术的不断发展，测试工具也得到了长足的发展，人们开始利用测试工具来帮助测试人员做一些重复性的工作。软件测试的一个显著特点就是其重复性，大量重复的工作使得工作量倍增，会很容易让人产生厌倦的心理，因此工具被用来解决这些问题。自动化测试就是使用自动化测试工具来代替手工进行的一系列测试动作，从而验证软件系统是否满足规定的需求或检测出预期结果与实际结果之间的差别。

1. 自动化测试

自动化测试的目的是减轻手工测试的工作量，以达到节约资源(包括人力、物力等)，保证软件质量，缩短测试周期的目的。通常是使用脚本或者其他代码驱动应用程序。这一切可以通过可视用户界面完成，也可以通过直接命令(如从客户端发向服务器，以模仿浏览器发送的命令)完成。

自动化测试，最容易联想到的是基于 GUI 录制回放的自动化功能测试工具，如 QTP、WinRunner 等；性能测试工具，如 LoadRunner、WepRunner 等；以及单元测试框架 JUnit 等。实际上自动化测试技术的含义非常广泛，任何帮助流程的自动流转，替换手工的动作、解决重复性问题以及大批量生产内容，从而帮助测试人员进行测试工作的相关技术或工具都叫做自动化测试技术。例如，用于辅助测试用例的设计或测试数据生成的测试用例设计工具。有些测试管理工具能帮助测试人员自动地统计测试结果并产生测试报告，编写脚本让版本编译自动进行。利用多线程技术去模拟并发请求，利用工具自动记录和监视程序的行为以及产生的数据，利用工具自动执行界面上的鼠标单击和键盘输入动作等。

2. 自动化测试的优点

通常软件测试的工作量很大，而测试中的许多操作是重复性的、非智力性和非创造性的，并要求做准确细致的工作，计算机就最适合代替人工去完成这样的任务。在过去的数年中，通过使用自动化的测试工具对软件的质量进行保障的例子已经数不胜数。目前为止自动化测试工具已经比较完善。测试人员可以通过在软件测试中应用自动化的测试工具来大幅度地提高软件测试的效率和质量。在决定采用自动化的测试工具之后，就应该尽早地开始测试的工作，这样可以使得修改错误变得更加的容易和廉价，并且可以减少更正错误后对软件开发周期的影响。

表 5.9 所示为国际软件质量保证组织的一个统计测试时间的对比，这个测试案例中包括 1750 个测试用例和 700 多个错误。

通过表 5.9 可以看出自动化测试与传统手工测试在各个阶段都有着很大的不同，自动化测试大约节省了将近 3/4 的工时，尤其是在测试执行和生成测试报告的方面，自动分析并生成测试反馈报告大量地节省了人工去收集结果并总结分析的过程，使得测试结果更加标准、准确、一致。另外，有很多测试是手工测试很难甚至是没办法做到的，不得不借助于工具的力量。例如，许多与时序、死锁、资源冲突、多线程等有关的错误，通过手工测试很难捕捉

到；进行系统负载、性能测试时，需要模拟大量数据或大量并发用户等各种应用场合时，很难通过手工测试来进行；进行系统可靠性测试时，需要模拟系统运行几年甚至几十年以验证系统能否稳定运行，这也是手工测试无法模拟的；如果有大量(如几千)的测试用例，需要在短时间内(如 1 天)完成，手工测试几乎不可能做到。

表 5.9　手工测试与自动化测试对比

测 试 步 骤	手工测试/h	自动化测试/h	通过使用工具的改善测试的百分比/%
测试计划	40	32	25
测试用例的开发	262	117	55
测试执行	466	23	95
测试结果分析	117	58	50
错误状态/更正监测	117	23	80
生成测试报告	96	16	83
时间总和	1090	277	75

测试自动化的优势是明显的，主要表现在以下几个方面。

(1) 测试自动化可以提高测试效率，使测试人员更加专注于新的测试模块的建立和开发，从而提高测试覆盖率；

(2) 测试自动化使测试资产的管理数字化，并使测试资产在整个软件测试生命周期内得以复用，这个特点在功能测试和回归测试中尤其具有意义；

(3) 通过测试流程的自动化管理提高了测试过程的有效性。

3. 自动化测试的局限

测试自动化带来诸多好处的同时，也带来了一些新的问题，主要表现在以下几个方面。

(1) 不能完全取代手工测试。自动化测试不可能也没必要取代手工测试来完成所有的测试任务。因为有些测试使用手工测试比自动化测试要简单，这时采用测试自动化的开销就比较大了。如果测试只是偶尔执行，或者待测系统经常变动、不稳定，测试需要大量的人工参与时，就不适宜采用自动化测试。

(2) 自动测试本身不具想象力。自动测试工具本身不具有想象力，只是按运行机制执行。而手工测试时，测试执行者可以在测试中判断测试输出是否正确，以及改进测试，还可以处理意外事件。如，网络连接中断时，必须重新建立连接。手工测试时可以及时处理该意外；而自动化测试时，该意外事件一般都会导致测试的中止。

(3) 对测试质量的依赖性较大。进行自动化测试实际上仅仅意味着测试的结果与期望值相同，因此测试的有效性很大程度上依赖于自动化测试本身的质量。确保测试的质量往往比自动化测试更为重要，所以要投入精力对测试软件进行必要的检测。

(4) 自动化测试成本可能高于手工测试。自动化测试的成本大致由以下几个部分组成：自动测试开发成本、运行成本、测试维护成本、工具成本和其他相关任务成本。

(5) 自动化测试可能会制约软件开发。应用软件的变化对自动化测试的影响要比手工测试更大一些，软件的部分改变有可能使自动化测试崩溃。而设计和实施自动化测试要比手工测试开销大，并需要维护。所以，对自动化测试影响较大的软件修改可能受到限制。

总地来说，自动化测试的优点和收益是显而易见的，但同时它也并非万能。只有正确地采用自动化测试并对其进行合理的设计和实施才能从中获益。

4. 自动化测试生命周期

自动化测试生命周期中包括六个阶段，分别是：决定自动化测试阶段；选择自动化测试工具；自动化测试的引入阶段；测试计划、设计和开发阶段；测试执行和管理阶段；测试项目评审阶段。

(1) 决定自动化测试阶段

在这一阶段，对于测试团队而言，重要的是进行测试计划的设计，知道自动化测试的预期结果，列出在正确执行自动化测试后的益处。同时，需要列出自动化测试工具的备选方案。以备对是否采用自动化测试进行准确评估，得出结论。

(2) 选择自动化测试工具

自动化测试工具的选择与采购是软件自动化测试生存周期方法学的第二个阶段，测试人员应该熟悉可用来支持整个生存周期各方面的不同类型的工具，能够对特定项目上实施的测试类型做出有益的决定。

自动化测试工具可以减少测试工作量，提高测试工作效率，但首先是能够选择合适的且满足企业系统工程环境的自动化测试工具。因为不同的测试工具，其面向的测试对象是不一样的，测试工具品种不一，功能性能差别较大，并且自动化测试工具的购置涉及资金、人员和资源等多方面的因素，是一项系统工程。一般来说，工具的选择首先是根据需求确定候选工具集，然后对工具的各个功能、价格等特性进行逐一的比较和综合评估，最后从候选工具中找出质量和性能、价格比等方面综合评分最佳的工具。对自动测试工具的适当选择，很大程度上决定了该工具能否带来相应的投资回报。

按照测试工具的主要用途和应用领域，测试工具分为测试用例设计工具、白盒测试工具、黑盒测试工具、性能测试工具、测试管理工具几个大类。

① 测试用例设计工具。测试用例设计工具一般用于辅助测试用例的设计或测试数据生成。常用的有 CTEXL、PICT 等。

② 白盒测试工具。白盒测试工具一般是针对程序内部逻辑流程和结果进行测试，测试中发现的缺陷可以定位到代码级。

根据测试工具原理的不同，白盒测试工具可以细分为静态测试工具和动态测试工具。静态测试工具直接对代码进行分析，不需要运行代码，也不需要对代码编译链接和生成可执行文件。静态测试工具一般是对代码进行语法扫描，找出不符合编码规范的地方，根据某种质量模型评价代码的质量，生成系统的调用关系图等。动态测试工具一般采用“插桩”的方式，在代码生成的可执行文件中插入一些监测代码，用来统计程序运行时的数据。它与静态测试工具最大的不同是动态静态工具要求被测系统实际运行。工具主要有开源的 XUnit，Parasoft 公司的 JTest、C++Test 等。

③ 功能测试工具。功能测试工具用于自动化执行功能测试脚本，一般采用录制回放的机制。通过录制操作过程，产生基本的测试脚本，然后在脚本编辑器中进一步地完善测试脚本，再通过回放脚本的方式执行脚本的测试步骤，从而实现功能测试的自动化。

功能测试工具能够有效地帮助测试人员对复杂的企业级应用的不同发布版本的功能进行测试，提高工作质量和效率。其主要目的是监测应用程序是否能够达到预期的功能并正常运

行。工具主要有 Mercury Interactive 公司的 WinRunner,IBM 公司的 Rational Robot 等。

④ 性能测试工具。性能测试工具的主要目的是度量应用系统的性能和可扩展性,是一种预测系统行为和性能的自动化测试工具。在实施并发负载过程中,通过实时性能监测来确认和查找问题,并针对所发现问题对系统进行优化,确保应用部署成功。性能测试工具能够对整个企业架构进行测试,通过这些测试,企业能最大限度地缩短测试时间,优化性能和加速应用系统的发布周期。工具主要有 Mercury Interactive 公司的 LoadRunner,RadView 公司的 WebLoad 等。

⑤ 测试管理工具。测试管理工具一般用于对测试需求、测试计划、测试用例、测试实施进行管理,测试管理工具还包括对缺陷的跟踪管理。测试管理工具能让测试人员、开发人员或其他的相关人员通过一个中央数据仓库在不同的地方就能交互信息。工具如 IBM 公司的 Rational TestSuite 系列中的 TestManager, Mercury Interactive 公司的 TestDireetor 等。

⑥ 测试辅助工具。这类工具需要针对不同的软件进行开发,其本身并不执行测试,而只是完成像生成测试数据为测试提供数据准备这样的工作。

(3) 自动化测试的引入阶段

自动化测试引入阶段是自动化测试生存周期方法学的第三个阶段,包括测试过程分析和测试工具考查。测试过程分析确保整个测试过程和策略适当,必要时可加以改进,以便成功地引入自动化测试。测试人员定义和收集测试过程度量以确保过程的改进。

在此阶段,确定使用的技术环境以及自动化工具可支持的各种测试。按照测试需求和计划中的测试活动,对用户参与计划进行评估并对测试组技能进行分析。强调测试人员的早期参与,支持将需求规范细化成能被充分测试的条款,并强化测试人员对应用程序需求与设计的了解。

测试工具的考查主要是综合考虑项目测试需求、可用的测试环境,测试平台、测试人员、用户环境以及被测应用系统的特性,测试人员应该研究引入自动化测试工具后是否对项目有益,还应该评审项目进度以确保为测试工具建立和需求体系留有足够的时间,将潜在的测试工具和实际应用程序映射到测试需求中,验证测试工具是否与应用和环境兼容,同时还应研究解决方案以解决兼容性防止测试期间出现的不兼容问题。

(4) 测试计划、设计和开发

测试计划、设计和开发是自动化测试生存周期方法学的第四个阶段,相对来说比较重要。测试计划是对软件测试过程的一个整体上的设计。在此阶段,通过收集项目和产品相关的信息,对测试范围、测试风险进行评估,对测试用例、工作量、支持测试环境所需的硬件、软件、网络和时间等进行评估,对测试采用的测试策略、方法、资源、环境、进度等做出合理的安排。在任何类型的测试中,测试计划都是必要的步骤。测试计划是进行成功测试的关键。任何类型测试的第一步都是制订比较详细的测试计划。好的测试计划能够保证测试工具完成测试的目标。

(5) 测试执行和管理

系统的需求分析和检查、测试计划以及测试用例的准备都是为了执行测试而准备的。在此阶段,测试团队必须根据测试的日常安排来执行测试脚本,并改善这些脚本,同时还必须评审测试的结果。系统的问题应通过系统问题报告一一记录在案,并帮助开发人员去理

解和重现这些问题。最后，测试团队还需要进行回归测试来追踪和关闭这些问题。

(6) 测试项目评审

测试项目的评审必须贯穿于整个自动化测试生命周期，以利于测试活动的不断改进，必须有相应的标准来衡量评审的结果。通过运用系统的方法，测试团队就能在测试资源受限的情况下利用这一途径组织和执行测试活动，并且达到使测试覆盖率最大的目的。

5.2.2 能力目标

了解软件测试自动化基本概念；掌握软件测试自动化的优、缺点；对自动化测试工具有初步的认识和了解；掌握软件测试自动化的生命周期。

5.2.3 任务驱动

1. 软件测试的自动化是否可以完全替代手工测试？试说明原因。
2. 软件测试的自动化流程有哪些？
3. 如何选择自动化测试工具？
4. 进行白盒测试的自动化工具有哪些？

5.2.4 实践环节

1. 通过网络，了解软件测试自动化发展的现状以及未来发展方向。
2. 查阅相关资料，说明 LoadRunner 是如何进行性能自动化测试。
3. 是否所有的测试阶段都适合软件测试的自动化？

5.3 小　　结

- 软件测试管理是一种活动，可以对各阶段的测试计划、测试案例、测试流程进行管理、跟踪，记录其结果，并将结果反馈给系统的开发者和管理者。同时将测试人员发现的错误立刻记录下来，生成问题报告并对之进行管理。
- 软件测试过程简单说就是围绕缺陷进行的，是对缺陷的跟踪管理。软件缺陷属性包括缺陷标识、缺陷类型、缺陷严重程度、缺陷产生可能性、缺陷优先级、缺陷状态、缺陷起源、缺陷来源、缺陷原因。
- 软件测试管理的主要功能包括：测试控制对象的编辑和管理，测试流程控制和管理，统计分析和决策支持。
- 自动化测试的目的是减轻手工测试的工作量，以达到节约资源(包括人力、物力等)，保证软件质量，缩短测试周期的目的。
- 自动化测试生命周期包括：决定自动化测试；选择自动化测试工具；自动化测试的引入阶段；测试计划、设计和开发阶段；测试执行和管理；测试项目评审阶段。
- 按照测试工具的主要用途和应用领域，测试工具分为测试用例设计工具、白盒测试工具、黑盒测试工具、性能测试工具、测试管理工具几个大类。

习　题　5

1. 软件测试管理内容包括哪些?
2. 简要阐述软件缺陷所包括的内容。
3. 软件测试自动化的优、缺点是什么?
4. 以下(　　)是手工测试的缺陷。
   A. 手工测试无法做到覆盖所有代码路径
   B. 许多与时序、死锁、资源冲突、多线程等有关的错误通过手工测试很难捕捉到
   C. 在系统负载、性能测试时,需要模拟大量数据或大量并发用户等各种应用场合时,也很难通过手工测试来进行
   D. 手工测试技术含量低
5. 以下(　　)不是自动化测试的优点。
   A. 节省时间,缩短周期　　B. 容易实施,结果可靠
   C. 做手工不能做的事情　　D. 不用人去测试
6. 以下(　　)是自动化测试的优点。
   A. 完成手工测试不能或难以完成的测试　　B. 测试不可重复
   C. 无法利用全部资源　　D. 需要人参与测试
7. 下面(　　)选项不是自动化测试产生的原因。
   A. 同样的测试需要执行多次　　B. 手工执行测试用例效率极低
   C. 人工执行测试很难模拟大量并发数据　　D. 需要人参与测试
8. 下面(　　)不是自动化测试的局限性所在。
   A. 不能完全取代手工测试和手工测试工程师
   B. 软件自动化测试可以降低测试的效率
   C. 如果缺乏测试经验,测试的组织差、文档少或不一致,则自动测试的效果比较差
   D. 软件自动化测试工具本身的问题
9. 下列有关测试过程管理的基本原则,(　　)是错误的。
   A. 测试过程管理应该首先建立测试计划
   B. 测试需求在测试过程中可以是模糊的、非完整的
   C. 在测试任务较多的情况下,应用建立测试任务的等级来优化处理
   D. 整个测试过程应该具有良好的可测性和可跟踪性,强调用数据说话
10. 为保证测试活动的可控性,必须在软件测试过程中进行软件测试配置管理,一般来说,软件测试配置管理中最基本的活动包括(　　)。
    A. 配置项标识、配置项控制、配置状态报告、配置审计
    B. 配置基线确立、配置项控制、配置报告、配置审计
    C. 配置项标识、配置项变更、配置审计、配置跟踪
    D. 配置项标识、配置项控制、配置状态报告、配置跟踪
11. 下列有关测试过程管理的基本原则,(　　)是正确的。
    A. 测试过程管理不需要首先建立测试计划

B. 测试需求在测试过程中可以是模糊的、非完整的
C. 在测试任务较多的情况下，应该按照难易来先后进行
D. 整个测试过程应该具有良好的可测性和可跟踪性，强调用数据说话

12. 下面(　　)不是软件测试工具。
A. JUnit　　B. WinRunner
C. QTP　　D. Tomcat

13. 下面(　　)是单元测试的测试工具。
A. JUnit　　B. WinRunner
C. QTP　　D. Tomcat

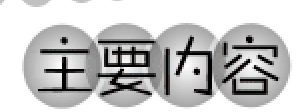
14. 下面(　　)是可以进行压力测试。
A. JUnit　　B. LoadRunner
C. QTP　　D. Tomcat

15. JUnit 中测试用例必须继承的父类是(　　)。
A. TestCase　　B. TestSuite
C. Test　　D. TestDemo

16. 属于白盒自动化测试工具的是(　　)。
A. JUnit　　B. LoadRunner
C. QTP　　D. WinRunner

第 6 章

QTP 测试工具



- QTP 的基本原理
- QTP 的应用

本章主要学习 QTP 的基本工作原理，以及应用 QTP 进行功能测试。QTP 是一个侧重于功能的回归自动化测试工具，采用关键字驱动的理念以简化测试用例的创建和维护。用户可以直接录制屏幕上的操作流程，自动生成功能测试或者回归测试用例。专业的测试者也可以通过提供的内置脚本和调试环境来取得对测试和对象属性的完全控制。重点掌握如何使用 QTP 创建测试脚本，增强测试脚本的功能，运行测试脚本并分析测试结果等。

6.1 QTP 的基本原理

6.1.1 核心知识

QuickTest Professional 简称 QTP，是 Mercury 公司基于关键字技术进行系统功能测试的最佳解决方案之一。通过 QTP 可以测试标准的 Windows Applications、Web Project、ActiveX 控件和 Visual Basic 应用程序。由于 QTP 兼容 Unicode 编码方式，从而使测试人员可以在多种国际语言环境下测试应用程序。

QTP 通过录制在应用程序中执行的操作，可以方便地创建测试和业务组件。测试通常是一个或多个操作的集合，用于验证应用程序是否按预期执行。业务组件用来对特定场景中的业务流程进行测试。

1. QTP 工作流程

当测试人员在浏览应用程序时，QTP 将录制执行的每个步骤，并生成在基于表的关键字视图中，用图形化的方式显示这些测试步骤。录制完成后，测试人员可以指示检查应用程序中特定对象的属性，并可以通过添加或修改关键字视图中的步骤来进一步增强测试的效果。这将有助于测试人员对应用程序或网站是否按预期工作进行监控。

QTP 工作流程如图 6.1 所示。

(1) 录制测试脚本

浏览应用程序或网站时，QTP 会将测试人员执行的每个步骤图形化显示为关键字视图



图 6.1　QTP 工作流程图

中的一行。关键字视图的“文档”列用通俗易懂的语句显示了每个步骤的描述。每个步骤都是引起网站或应用程序发生更改的事件。例如，单击链接或图像，或者提交数据表单等。

（2）插入检查点

检查点用来检查页面、对象或文本字符串中的特定值或属性，通过它可以标识网站或应用程序是否正常运行。

（3）测试脚本参数化

测试网站或应用程序时，可以参数化测试脚本以检查应用程序如何使用不同数据执行相同的操作。测试人员可以提供数据表中的数据，定义环境变量，定义测试组件，操作参数，生成随机数字等。参数化测试脚本时，QTP 将用参数代替测试脚本中的固定值。当使用数据表参数时，测试工具将采用循环的方式使用数据表不同行中的值。

（4）运行测试脚本

测试脚本将从其第一行开始运行直至测试结束时停止。在运行中，QTP 将连接到待测网站或应用程序，执行测试脚本中的每一项操作，检查所有指定的文本字符串、对象或表。如果使用数据表参数对测试进行了参数化，测试工具还将对定义的每组数据值重复执行该测试。

（5）测试结果分析

运行测试脚本之后，可以在“测试结果”窗口中查看运行的结果。测试人员既可以查看结果的概要，又可以查看详细报告。

2. QTP 测试工具的安装

本教材中选用业界应用较为成熟的版本 QTP9.0。安装 QTP9.0 要求内存至少 512MB，硬盘空间 500MB。具体的安装步骤如下。

（1）进入欢迎页面。选择“QuickTest Professional 安装程序”选项，如图 6.2 所示。



图 6.2　QTP 欢迎页面

（2）阅读相关软件许可协议。选择“我接受该许可证协议中的条款”选项，单击“是”按钮。

（3）许可证类型。选择“演示版”选项，单击“下一步”按钮。

（4）启用 QTP 远程执行。选择“自动设置这些选项”，单击“下一步”按钮。

（5）调试支持。单击“下一步”按钮。

（6）安装类型。选择“完全”选项，并指定安装目录，然后单击“完成”按钮。

安装完成后，选择“开始”→“程序”→QuickTest Professional→QuickTest Professional 命令，将打开如图 6.3 所示对话框。

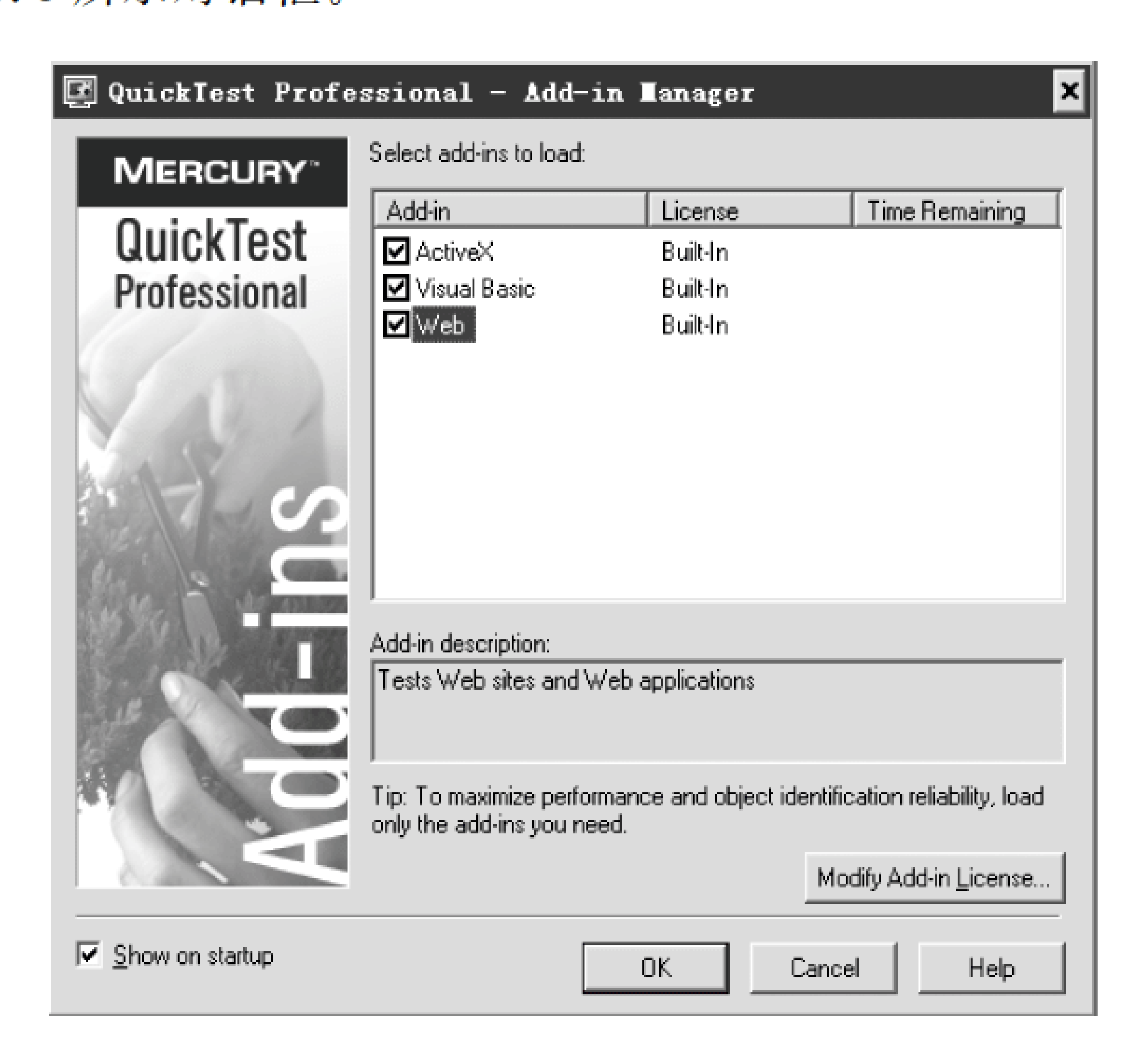


图 6.3　插件选择页面

在 QTP 启动时，可以选择支持的插件，有 ActiveX、Visual Basic 和 Web。这里可以全部选上，单击 OK 按钮，计入主页面。

主页面包括：测试工具条、Debug 工具条、测试脚本管理窗口、Data Table 窗口和 Active Screen 窗口等，如图 6.4 所示。

- 测试工具条。包含了在创建、管理测试脚本时要使用的按钮，如图 6.5 所示。
- 调试工具条。包含在调试测试脚本时要使用的工具条，如图 6.6 所示。
- 测试脚本管理窗口。提供了两个可切换的窗口，分别通过图形化方式和 VBScript 脚本方式来管理测试脚本。
- Data Table 窗口。用于参数化测试脚本。
- Active Screen 窗口。当录制会话过程中执行某个特定步骤时，Active Screen 窗口提供了应用程序的快照。此外 Active Screen 窗口中显示的内容可以包含该页中每个对象的详细属性信息。

3. 测试运行环境

测试运行环境将基于窗体应用程序进行功能测试，项目来自于 QTP 附带的机票预订服务系统 Flight。通过该系统用户可以连接到数据库服务器，完成搜索航班、预订航班并查看航班路线等功能。

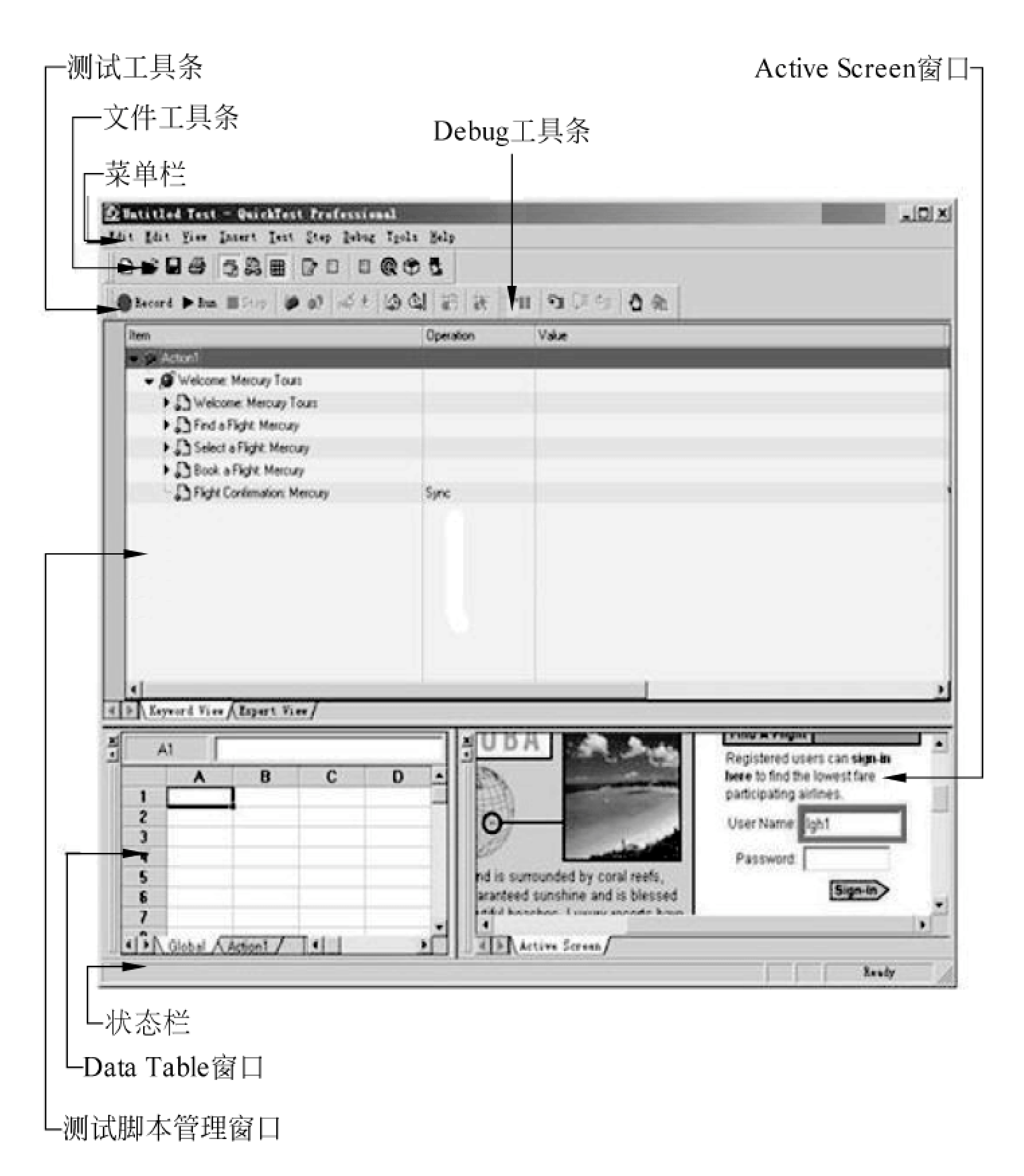


图 6.4　QTP 主页面

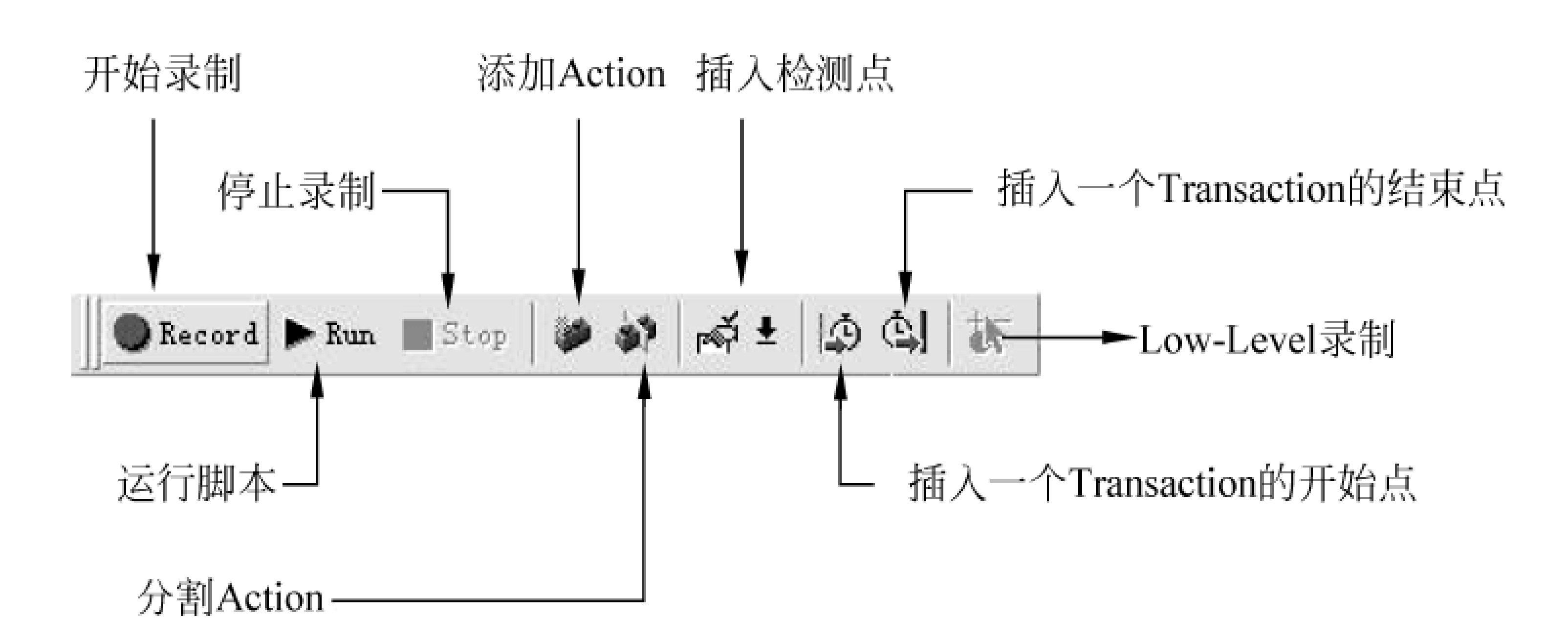


图 6.5　测试工具条

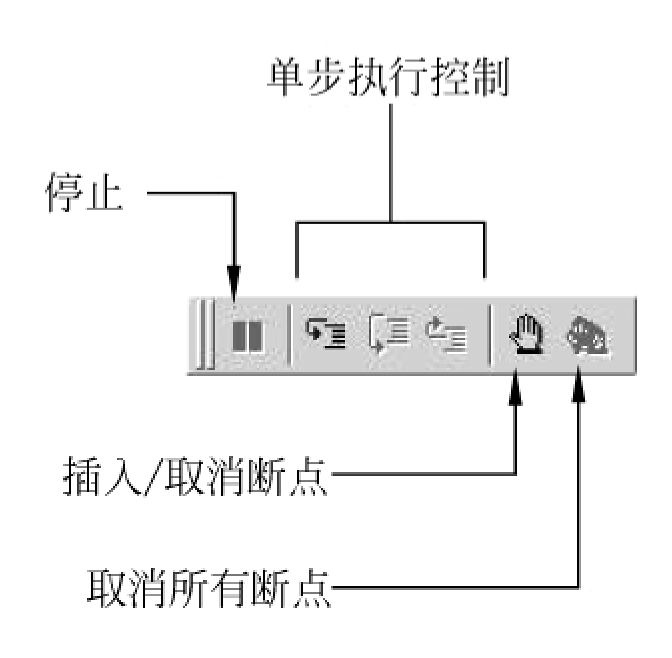


图 6.6　调试工具条

启动测试运行环境的方法为：选择“开始”→“程序”→QuickTest Professional→Sample Applications→Flight 命令，启动页面如图 6.7 所示。

使用任意用户名（要求长度至少 4 个字符），密码为 Mercury。

(1) 录制测试脚本

① 启动 QTP。选择“开始”→“程序”→QuickTest Professional→QuickTest Professional 命令。

② 启动机票预订服务系统 Flight 并登录。

注意：调整 QTP 与待测系统窗口的大小和位置，尽量使两个窗口不重叠。

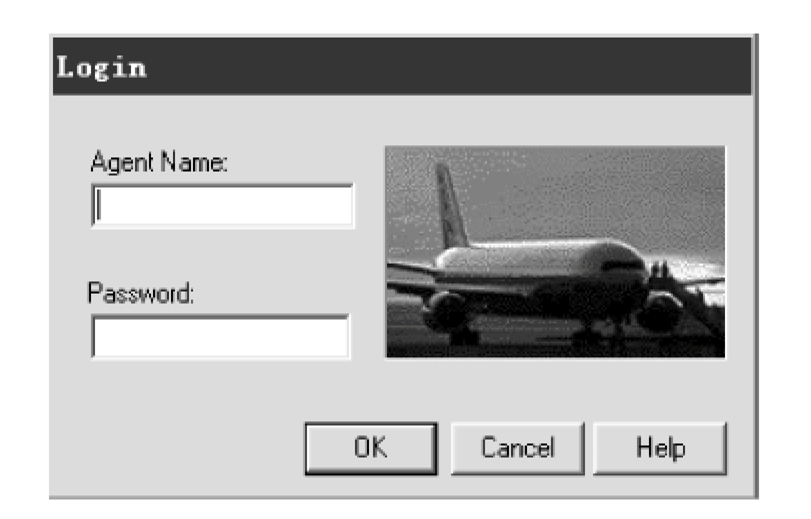


图 6.7 待测系统启动页面

③ 录制测试脚本。单击测试工具条中的 Record 按钮，打开 Record and Run Settings 对话框，因为待测系统为窗体应用程序，所以选择 Windows Applications 选项卡，如图 6.8 所示。

单击绿色＋按钮，打开一个 Application Details 对话框，如图 6.9 所示。



图 6.8 运行时设置

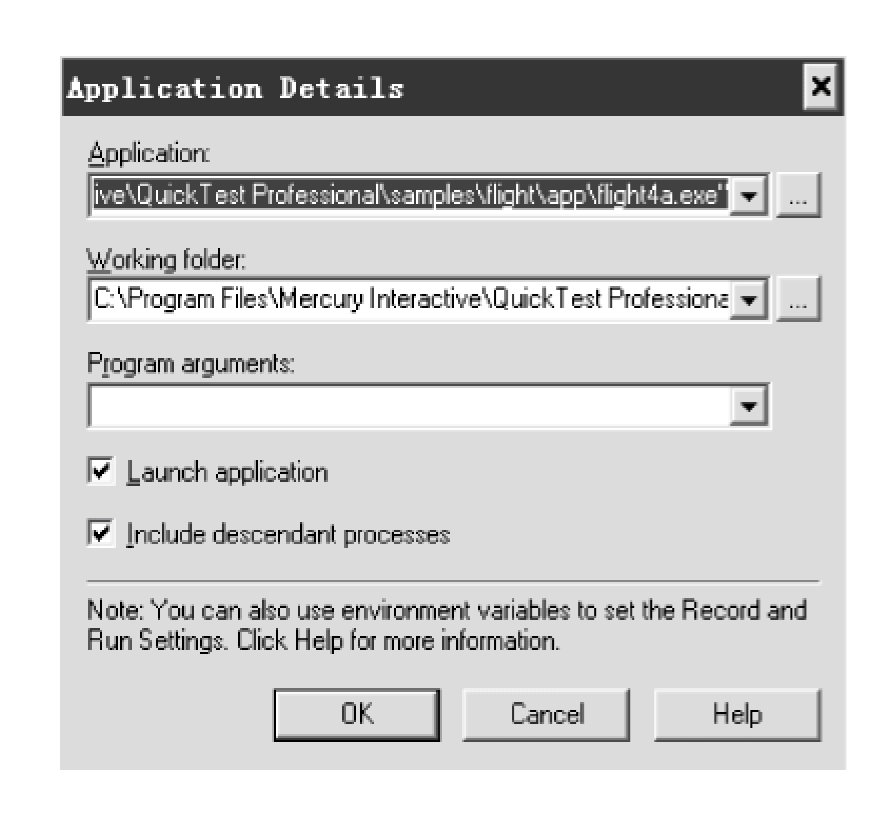


图 6.9 应用细节设计页面

在该对话框中不需要做任何修改，单击 OK 按钮。窗体将返回到 Record and Run Settings 对话框，单击“确定”按钮，开始录制测试脚本。

首先使用用户名 Mercury 和密码 Mercury 登录机票预订服务系统 Flight，接着在菜单中选择 File→Open Order 命令，在 Open Order 对话框中勾选 Order No. 复选框并在其文本框中录入“1”，表示打开第一个订单，最后在菜单中选择 File→Exit 命令退出。

(2) 停止录制脚本

单击测试工具条中的 Stop 按钮，停止录制测试脚本。

(3) 保存测试脚本

选择菜单栏中的 File→Save 命令，保存的测试脚本名为 Test1。

(4) 分析测试脚本

在录制过程中，QTP 在测试脚本管理窗口中产生对每一个操作的相应记录，并在

Keyword View 页面中以类似 Excel 工作表的方式显示所录制的测试脚本。当录制结束后，QTP 就记录下了测试过程中的所有操作。测试脚本管理窗口显示的内容如图 6.10 所示。

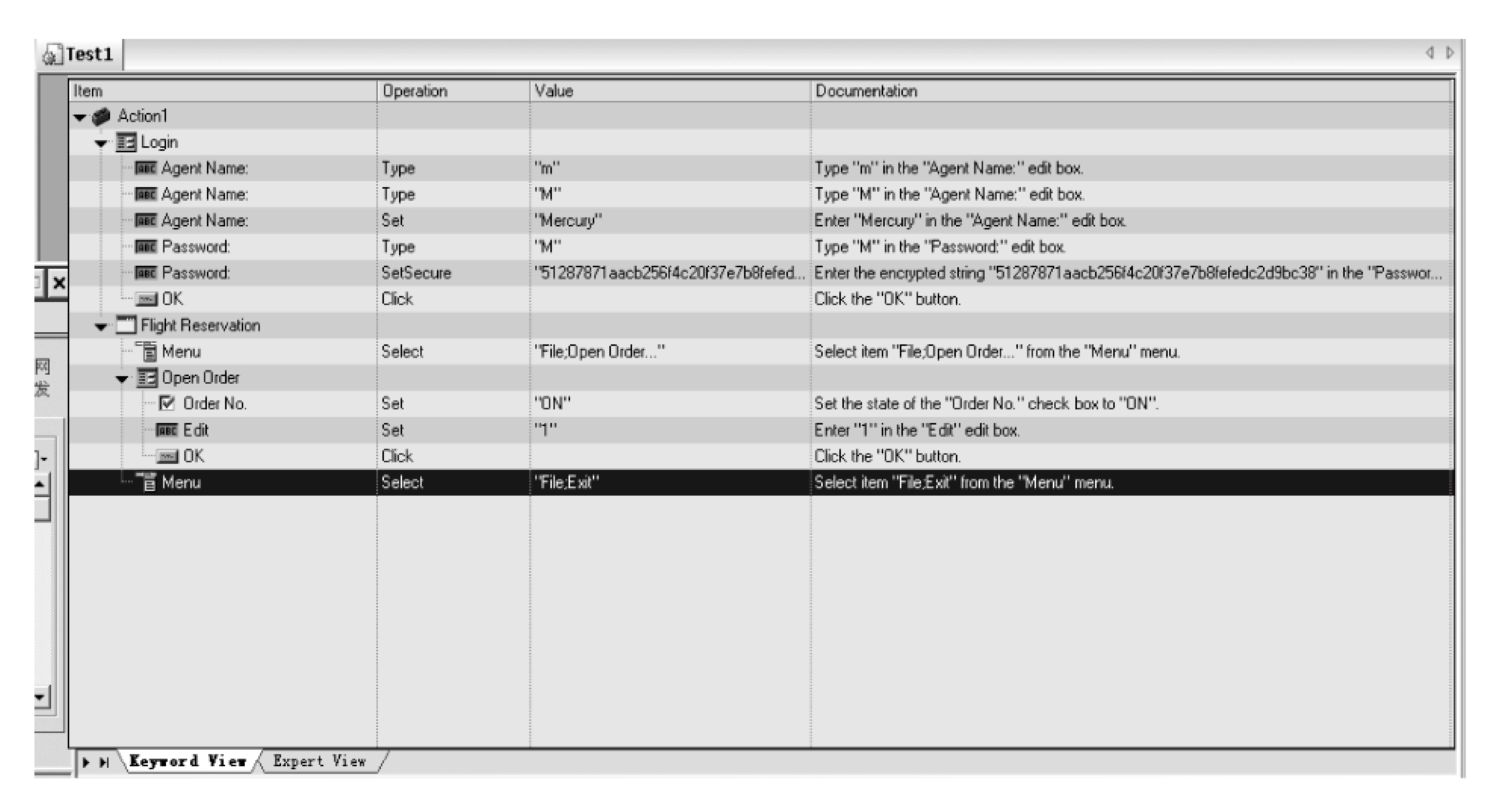


图 6.10 测试脚本管理窗口

在 Keyword View 页面中的每一个字段都有其特定的含义，具体说明如下。

- Item：以阶层式的图标表示这个操作步骤所作用的组件（测试对象、工具对象、函数或脚本）。
- Operation：在组件上执行的动作。如单击、选择等。
- Value：执行动作时的参数。例如，当鼠标单击一张图片时是用左键还是右键。
- Documentation：自动产生用来描述操作步骤的英文说明。

脚本中的每一个步骤在 Keyword View 页面中都会以一列来显示，用来表示此组件类别的图标以及操作步骤的详细数据。下面针对一些常见的操作步骤作详细说明，如表 6.1 所示。

表 6.1 组件说明

步　骤	说　明
Action1	一个动作的名称
Login	Login 为登录窗体
Agent Name: Set "Mercury"	Agent Name 是 edit box（文本框）的名称； Set 是在这个 edit box 上执行的动作； Mercury 是被输入的值
Password: SetSecure "51287871aacb256f4c20f37e7b8fefed...	Password 是 edit box 的名称； SetSecure 是在这个 edit box 上执行的动作，此动作有加密的功能； 51287871…是加密后的密码

续表

步　　骤	说　　明
Menu　Select　"File;Open Order..."	Menu 是菜单的名称； Select 是菜单上执行的动作； File；Open Order 代表在菜单中选择打开一个订单
Order No.　Set　"ON"	OrderNo 是 check box（复选框）的名称； Set 是在这个 edit box 上执行的动作； ON 表示复选框被选中
Edit　Set　"1"	Edit 是 edit box(文本框)的名称； Set 是在这个 edit box 上执行的动作； 1 是被输入的值
OK　Click	OK 是 button(按钮)的名称； Click 是 button 按下的动作
Menu　Select　"File;Exit"	Menu 是菜单的名称； Select 是菜单上执行的动作； File；Exit 代表系统退出

（5）执行测试脚本

当运行录制好的测试脚本时，QTP 会自动打开被测试程序，执行测试人员在测试中录制的每一个操作。测试运行结束后，会显示本次运行的结果。接下来，执行录制好的测试脚本 Test1。

① 打开测试脚本 Test1。在 QTP 菜单中选择 File→Open→Test 命令，浏览测试脚本的位置，单击“打开”按钮。

② 设置运行选项。在菜单中选择 Tool→Options 命令打开设置选项对话框，选择 Run 选项卡，如图 6.11 所示。

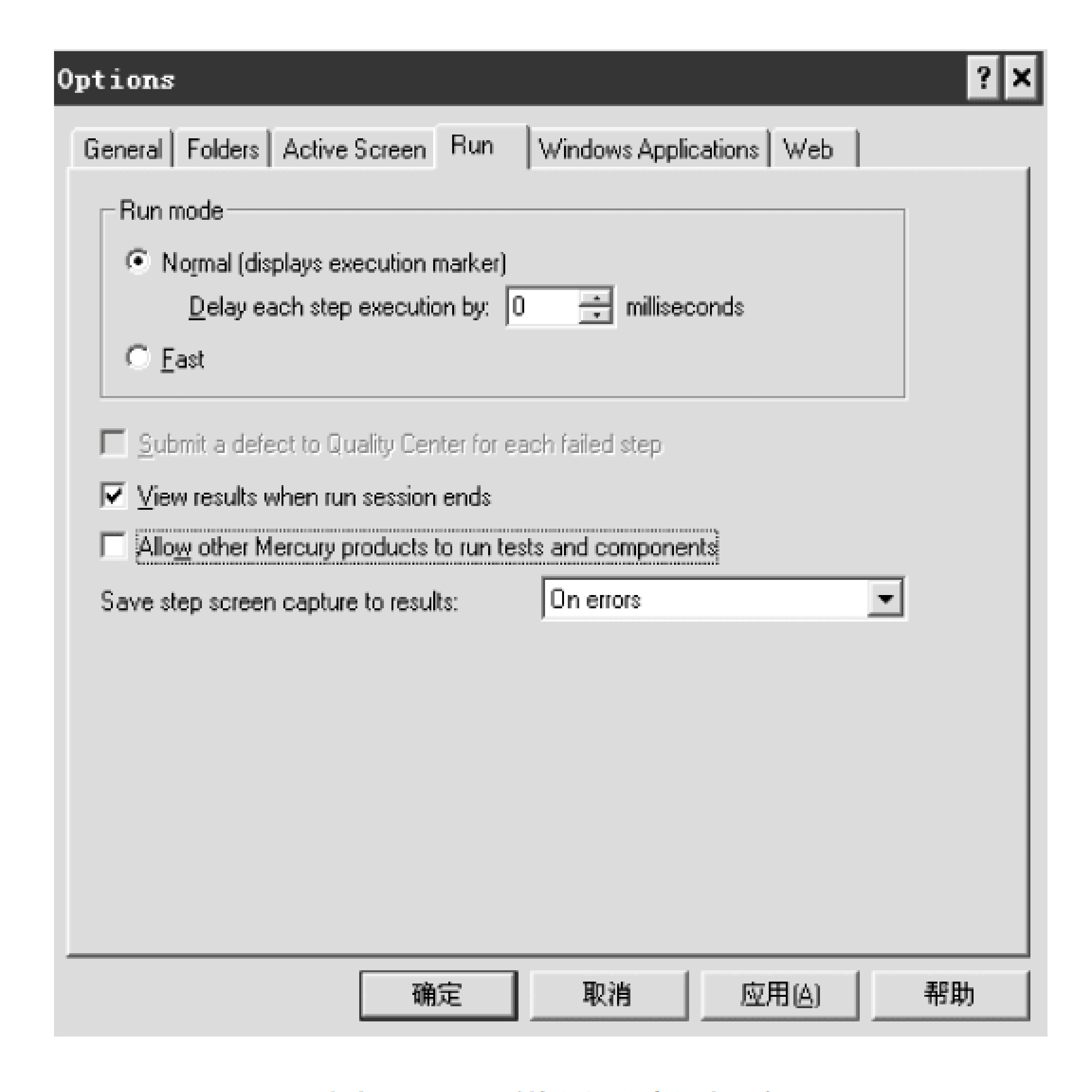


图 6.11　设置运行选项

如果要将所有画面储存在测试结果中,则在 Save step screen capture to results 下拉列表框中选择 Always 选项。一般情况下选择 On errors 或 On error and warning 表示在回放测试过程中出现问题时,才保存图像信息。在这里为了更多地展示 QTP 的功能,所以选择使用 Always 选项。

③ 在测试工具条中单击 Run 按钮,将打开 Run 对话框,指定测试结果保存路径和名称,这里使用系统默认值,如图 6.12 所示。

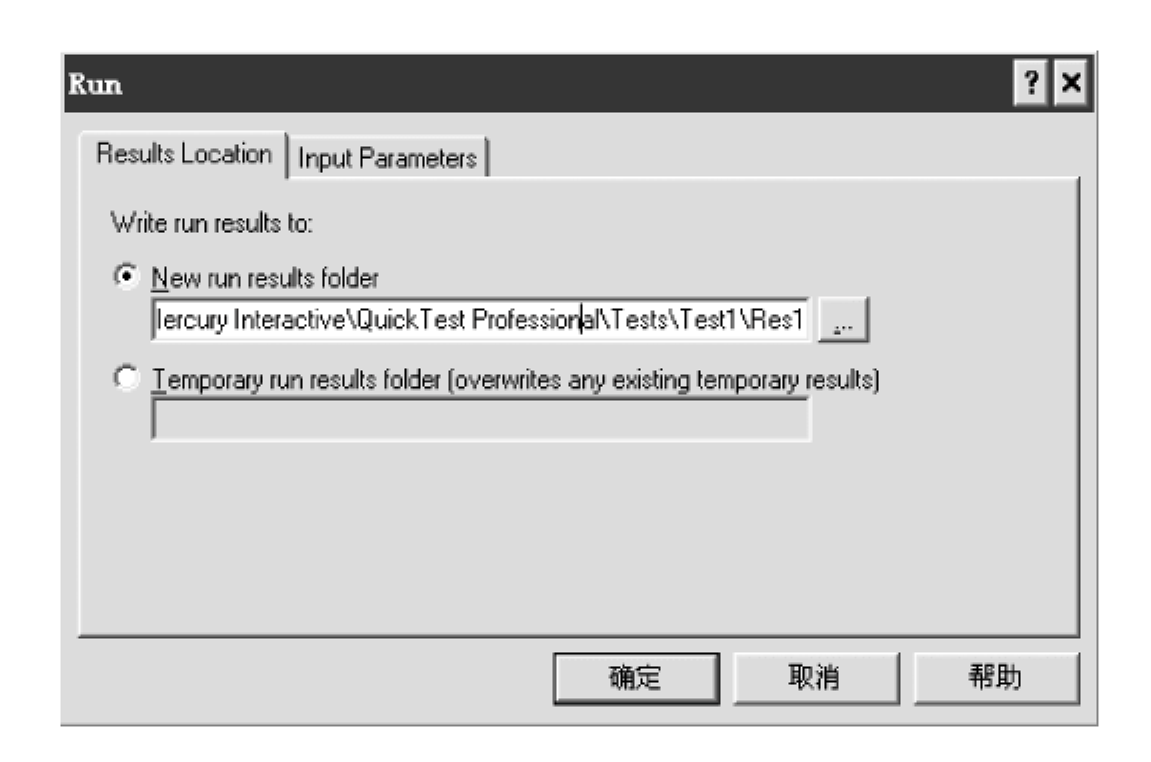


图 6.12　测试结果保存路径

④ 单击“确定”按钮开始执行测试。

可以看到 QTP 按照测试脚本中录制的操作,一步一步地运行测试,操作过程与手工操作时完全一样。同时可以在 QTP 的 Keyword View 页面中会出现一个黄色的箭头,指示目前正在执行的测试步骤。

(6) 分析测试结果

在测试执行完成后,QTP 会自动显示测试结果窗口,如图 6.13 所示。

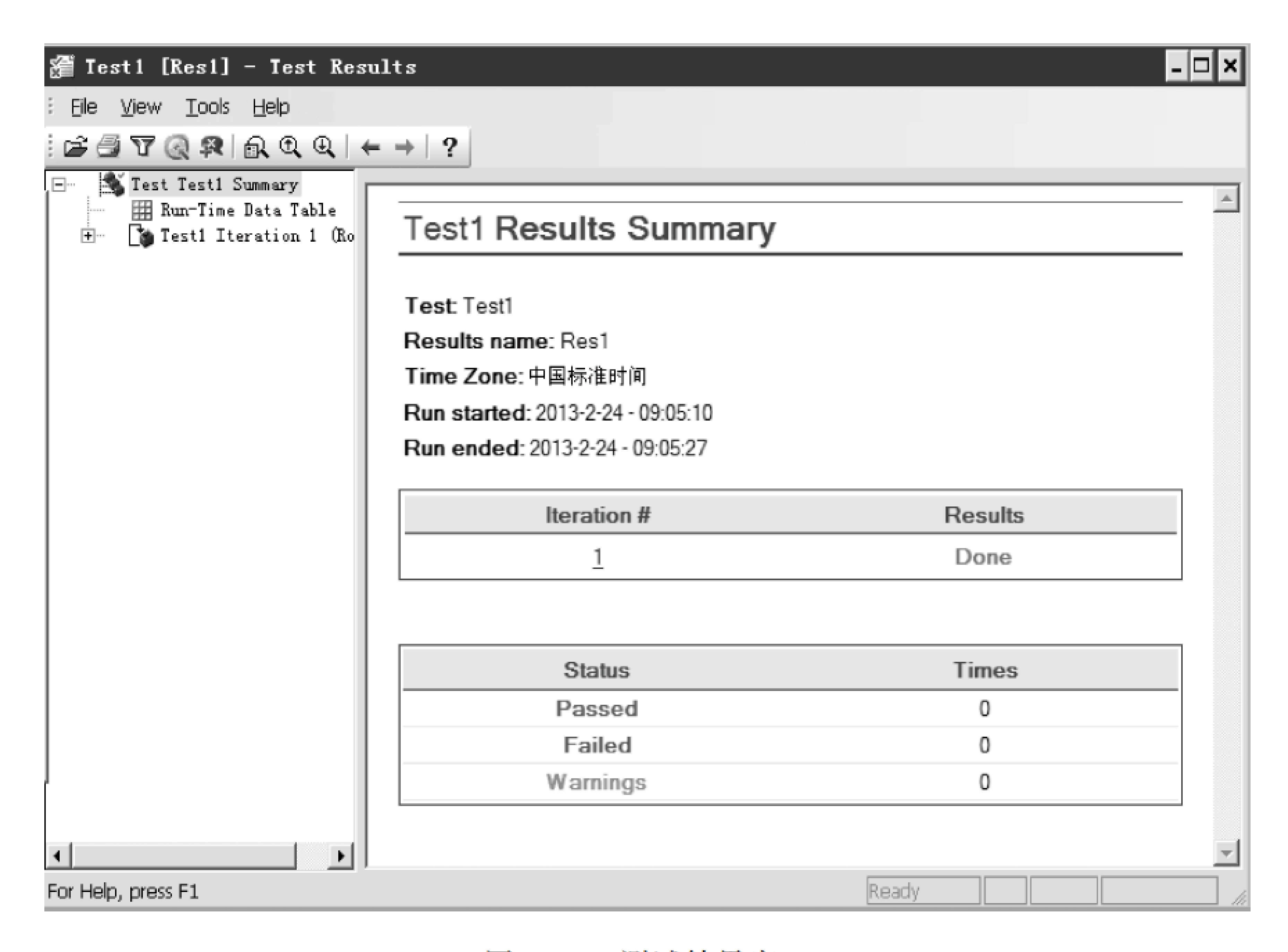


图 6.13　测试结果窗口

测试结果窗口中分两个部分显示测试执行的结果。

- 左侧为 Test Results 树视图,以阶层图标的方式显示测试脚本所执行的步骤。可以选择+图标检查每一个步骤,所有的执行步骤都会以图例的方式显示。可以对测试脚本中的某个或几个动作设置反复执行,每执行一次反复称为一个迭代,每一次迭代都会被编号。在本次测试脚本中,因为没有设置,故迭代次数为 1。
- 右侧显示测试结果的详细信息。在第一个表格中显示哪些迭代是已经通过的,哪些是失败的。第二个表格是显示测试脚本的检查点,哪些是通过的,哪些是失败的,以及有几个警告信息。

在上面的测试中,所有的测试都是通过的,在脚本中也没有添加检查点(有关检查点的内容将在下节中学习)。

为了能够查看执行测试脚本的详细结果,可以选择某个测试步骤进行显示。只需要选择+图标展开对应的测试步骤,就可以看到详细的脚本执行结果。

6.1.2 能力目标

了解 QTP 测试工具的测试重点;掌握 QTP 的测试流程;能够成功安装 QTP 软件;了解 QTP 软件的测试页面以及运行测试环境。

6.1.3 任务驱动

1. QTP 是针对什么方面的自动化测试工具?测试重点是什么?
2. QTP 的测试流程包括几个步骤?
3. 录制用户操作脚本包括哪些内容?

6.1.4 实践环节

1. 安装 QTP 软件和机票预订服务系统 Flight。

2. 录制测试脚本 TestCase,其中包括用户登录、查找航线、预订机票、订单确认、退出系统等功能。执行脚本,得到测试分析报告。

6.2 QTP 的应用

6.2.1 核心知识

通过上一节的学习,已经掌握了如何录制、执行测试脚本以及查看测试结果。但是只是实现了测试执行的自动化,没有实现测试验证的自动化,所以这并不是真正的自动化测试。在本节中将在测试脚本中设置检查点,以验证执行结果的正确性。

1. 检查点

检查点是将指定属性的当前值与该属性的期望值进行比较的验证点。这能够确定网站或应用程序是否正常运行。当添加检查点时,QTP 会将检查点添加到关键字视图中的当前行,并在专家视图中添加一条"检查检查点"语句。运行测试或组件时,QTP 会将检查点的期望结果与当前结果进行比较。如果结果不匹配,检查点就会失败。可以在"测试结果"窗口中查看检查点的结果。

QTP 支持的检查点种类，如表 6.2 所示。

表 6.2 QTP 检查点种类表

检查点类型	说　明	示　例
标准检查点	检查对象的属性	检查某个按钮是否被选取
图片检查点	检查图片的属性	检查图片的来源文件是否正确
表格检查点	检查表格的内容	检查表格的内容是否正确
网页检查点	检查网页的属性	检查网页加载的时间或是网页是否含有不正确的链接
文字/文字区域检查点	检查网页上或是窗口上出现的文字是否正确	检查登录系统后是否出现登录成功的文字
图像检查点	提取网页和窗口的画面检查画面是否正确	检查网页或者网页的一部分是否如期显示
数据库检查点	检查数据库的内容是否正确	检查数据库查询的值是否正确
XML 检查点	检查 XML 文件的内容	XML 检测点有两种——XML 文件检测点和 XML 应用检测点。XML 文件检测点用于检查一个 XML 文件；XML 应用检测点用于检查一个 Web 页面的 XML 文档

比较常用的检查点有：标准检查点、网页检查点、文字/文字区域检查点以及表格检查点。下面分别在测试脚本上建立这几种检查点。

QTP 除了提供基于窗体应用程序待测系统外，同时提供一个 Web 待测系统。启动 Web 待测系统的方法为：选择“开始”→“程序”→QuickTest Professional→Sample Applications→Mercury Tours Web Site 命令。

录制测试脚本的方式与录制窗体应用程序的方式相同，这里不再重复。这里录制测试脚本 Test2，其中包括用户登录、查找航线、预订机票、订单确认、退出系统等功能。

在 Test2 测试脚本中创建 4 个检查点，分别是：标准检查点、网页检查点、文字/文字区域检查点以及表格检查点。

(1) 标准检查点

通过向测试或组件中添加标准检查点，可以对不同版本的应用程序或网站中的对象属性值进行比较。可以使用标准检查点来检查网站或应用程序中的对象属性值。标准检查点将对录制期间捕获的对象属性的预期值，与运行会话期间对象的当前值进行比较。

首先在 Test2 测试脚本上添加一个标准检查点，这个检查点用以检查旅客的姓名。创建标准检查点的方法如下。

① 打开 Test2 测试脚本。

② 选择要建立检查点的网页。在 QTP 的视图树中展开 Action1→Welcome：Mercury Tours→Book a Flight：Mercury 目录，由于输入使用者姓名的测试步骤是 passFirst0 这个步骤，所以要选择这个步骤的下一个测试步骤，以便建立检查点，如图 6.14 所示。

③ 建立标准检查点。在 Active Screen 窗口的 First Name 文本框中右击，在弹出的菜单中显示插入选择点的类型，如图 6.15 所示。

选择 Insert Standard Checkpoint 选项，显示 Object Selection-Checkpoint Properties 对话框，如图 6.16 所示。

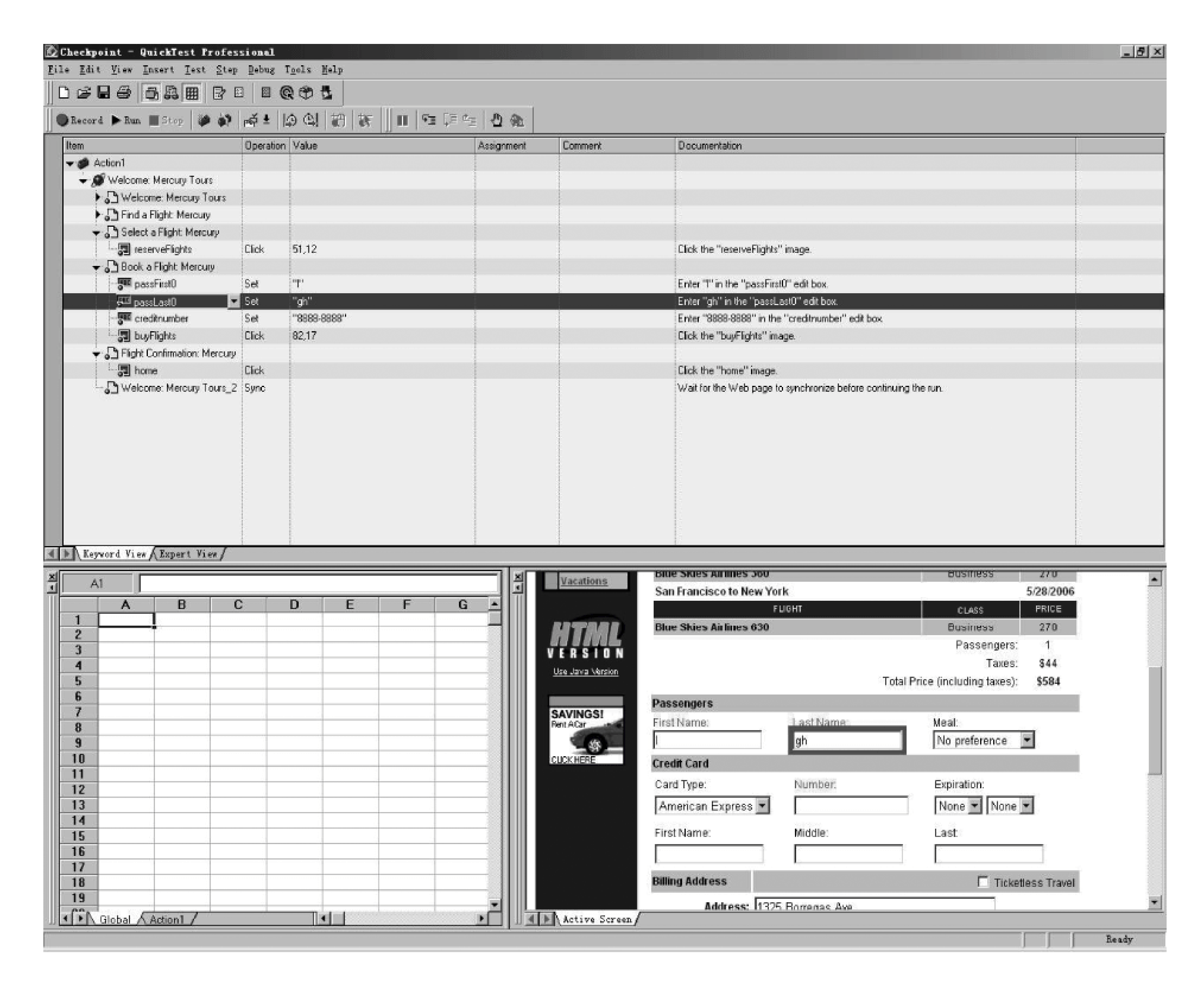


图 6.14　Test2 测试脚本图

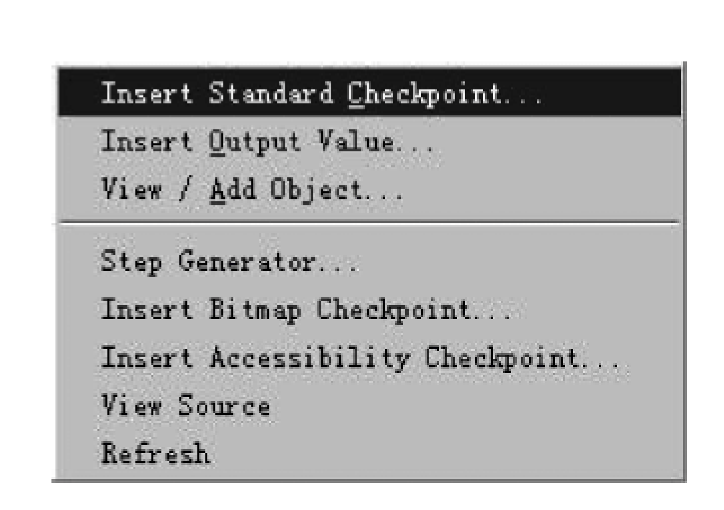


图 6.15　插入标准检查点

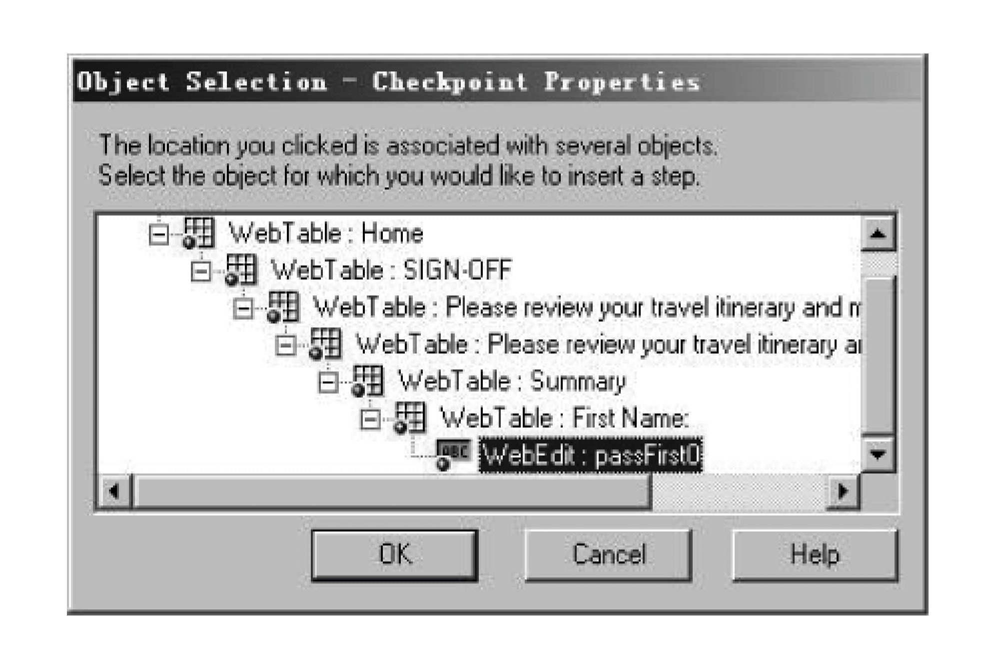


图 6.16　脚本对象属性

确保当前的焦点定位在 WebEdit：passFirst0 选项上，单击 OK 按钮，弹出如图 6.17 所示对话框。

在检查点属性窗口会显示检查点的属性如下。

- Name：检查点的名称。
- Class：检查点的类别，WebEdit 表示这个检查点是个输入框。
- Type 字段中的 ABC 图标：表示这个属性的值是一个常数。

对于每一个检查点，QTP 会使用预设的属性作为检查点的属性，表 6.3 说明了这些预设属性的含义。

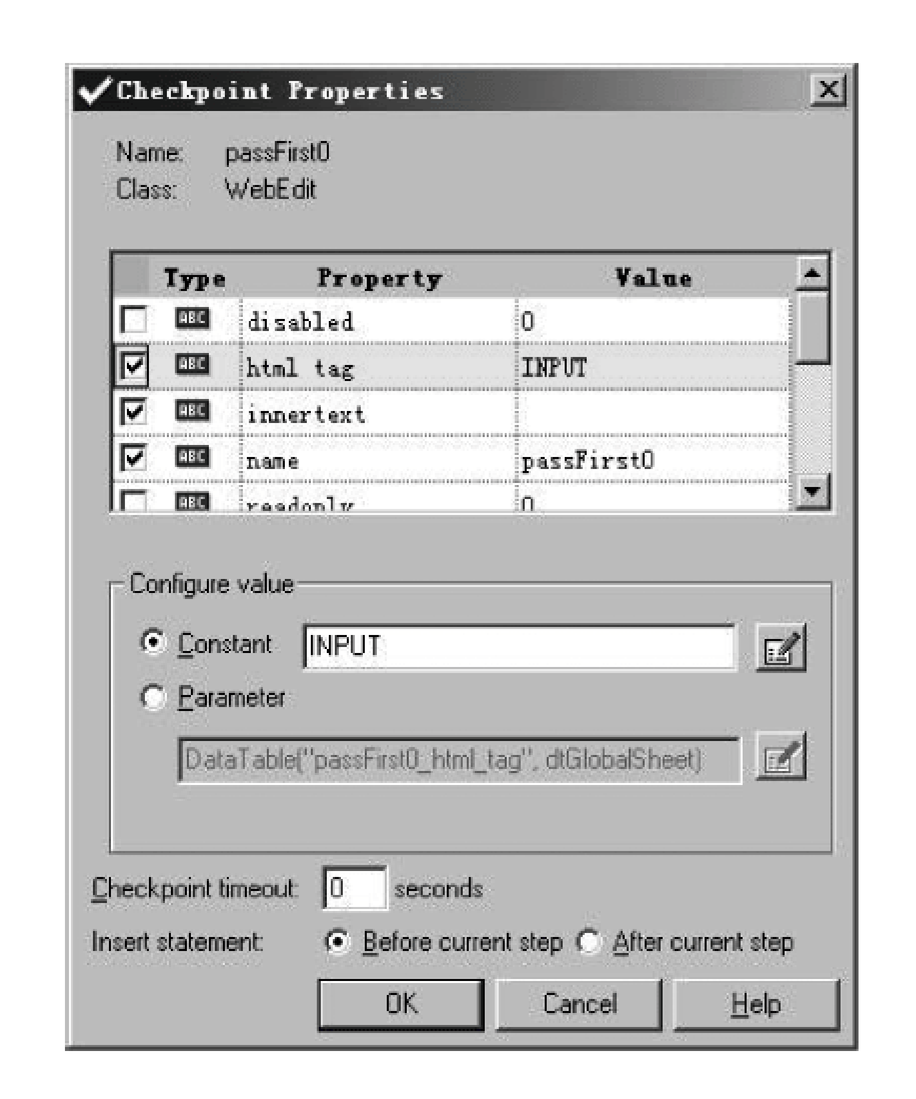


图 6.17 检查点属性设置

表 6.3 预设检查点属性

属 性	值	说 明
html tag	INPUT	HTML 原始码中的 INPUT 标签
innertext		在本例中，innertext 为空，检查点会检查当执行时这个属性是不是为空
name	passFirst0	passFirst0 是这个编辑框的名称
type	text	text 是 HTML 原始码中 INPUT 对象的类型
value	姓名	在编辑框中输入的文字

接受预设的设定值，单击 OK 按钮。QTP 会在选取的步骤之前建立一个标准检查点，如图 6.18 所示。



图 6.18 设置检查点脚本

④ 在工具栏上单击 Save 按钮保存脚本。通过以上 4 个步骤，完成了一个标准检查点的添加操作。

(2) 网页检查点

网页检查点会检查网页的链接以及图像的数量，与当前录制时的数量比较是否一致。网页检查点只能应用于 Web 项目中。在 Test2 测试脚本中再添加一个网页检查点，创建网页检查的步骤如下。

① 选择要建立检查点的网页。展开 Action1→Welcome：Mercury Tours 目录，选择 Book a Flight：Mercury 选项，在 Active Screen 窗口会显示相应的页面。

② 建立网页检查点。在 Active Screen 中的任意地方右击，选择 Insert Standard Checkpoint 命令，打开 Object Selection-Checkpoint Properties 对话框(由于选择的位置不同，对话框显示被选取的对象可能不一样)，如图 6.19 所示。

选择最上面的 Page：Book a Flight：Mercury 选项，并单击 OK 按钮确认，将打开 Page Checkpoint Properties 对话框，如图 6.20 所示。

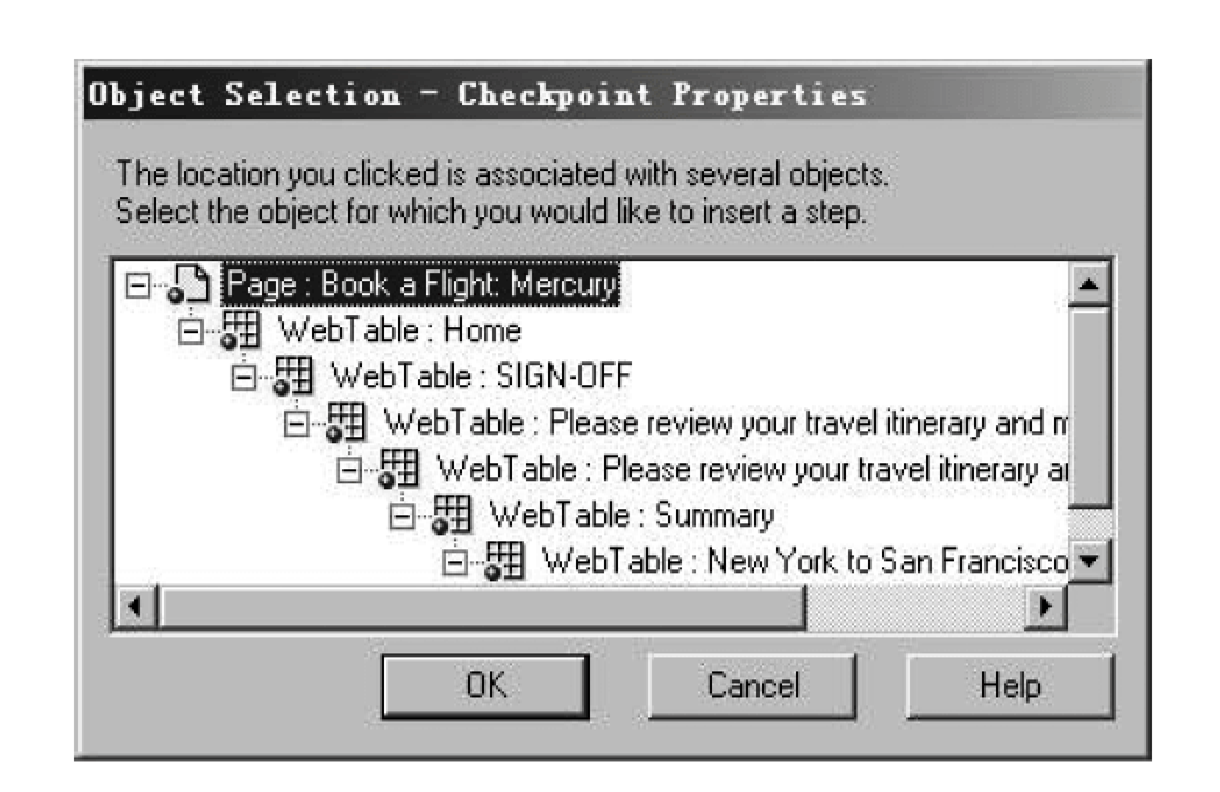


图 6.19　脚本对象属性

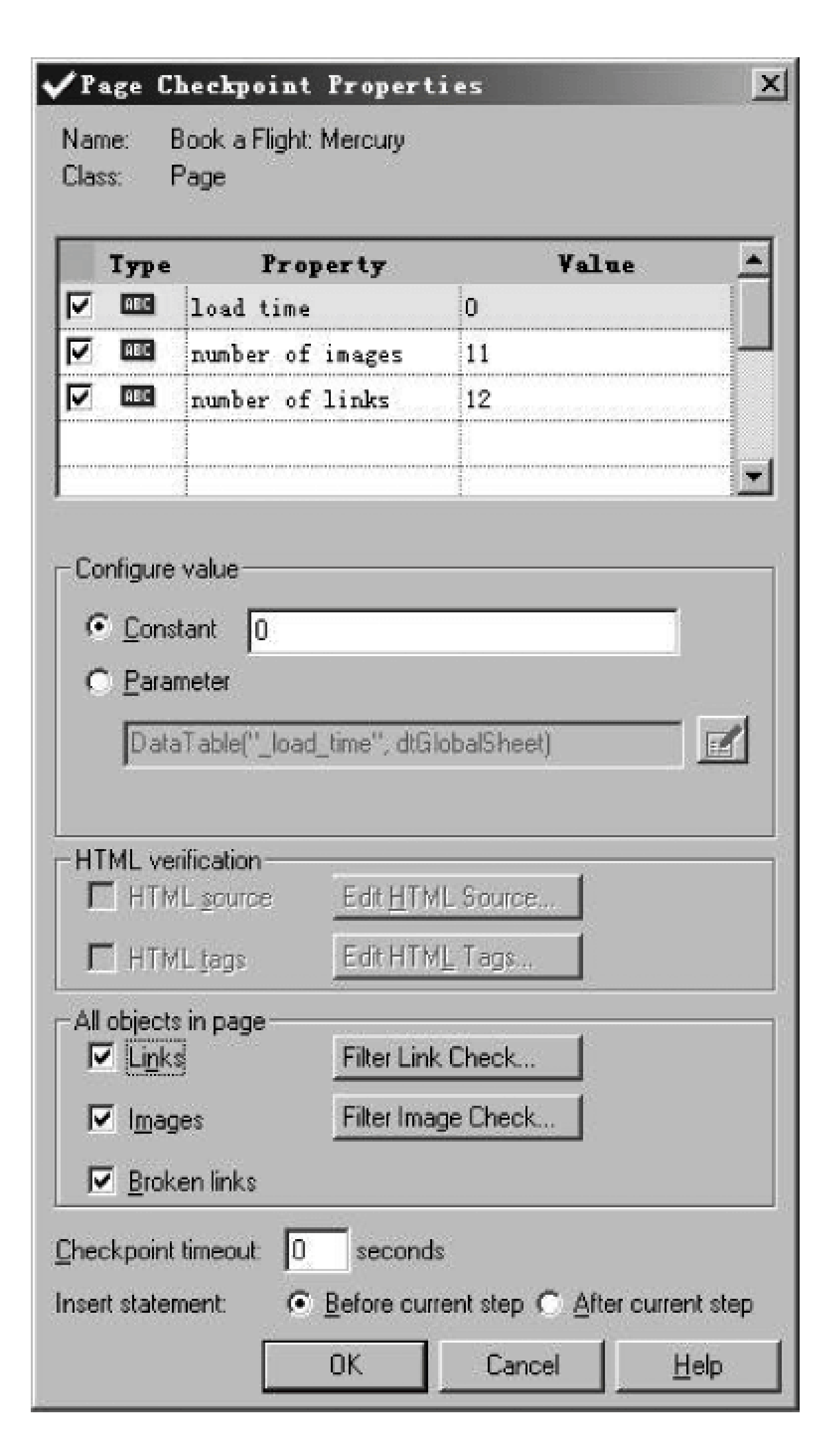


图 6.20　网页检查点属性设置

当执行测试时，QTP 会检查网页的链接与图片的数量，以及加载的时间，是否和对话窗口上方所显示的内容相同。QTP 检查每一个链接的 URL 以及每一个图片的原始文件是否存在。接受默认设定，单击 OK 按钮。QTP 会在 Book a Flight：Mercury 网页上加一个网页检查。

③ 在工具栏上单击 Save 按钮保存脚本。

(3) 文字/文字区域检查点

建立一个文字检查点，检查在 Flight Confirmation 网页中是否出现"New York"文字。具体的执行步骤如下。

① 确定要建立检查点的网页。展开 Action1→Welcome：Mercury Tours 目录，选择 Flight Confirmation：Mercury 选项，在 Active Screen 窗口会显示相应的页面。

② 建立文字检查点。在 Active Screen 窗口中选择 Departing 下方的"New York"文字。

在选取的文字上右击，并选择弹出菜单中的 Insert Text Checkpoint 命令，打开 Text Checkpoint Properties 对话框，如图 6.21 所示。

当 Checked Text 出现在下拉列表框中时，在 Constant 文本框中显示的就是选取的文字，这也就是 QTP 在执行测试脚本时所要检查的文字。

③ 单击 OK 按钮关闭窗口。QTP 会在测试脚本中加上一个文字检查点，这个文字检查点会出现在 Flight Confirmation：Mercury 网页下方。

④ 在工具栏上单击 Save 按钮保存脚本。

(4) 表格检查点

通过向测试或组件中添加表格检查点，可以检查表的单元格中是否显示了指定的值。对于 ActiveX 表，还可以检查表对象的属性。要添加表格检查点，可使用“检查点属性”对话框。

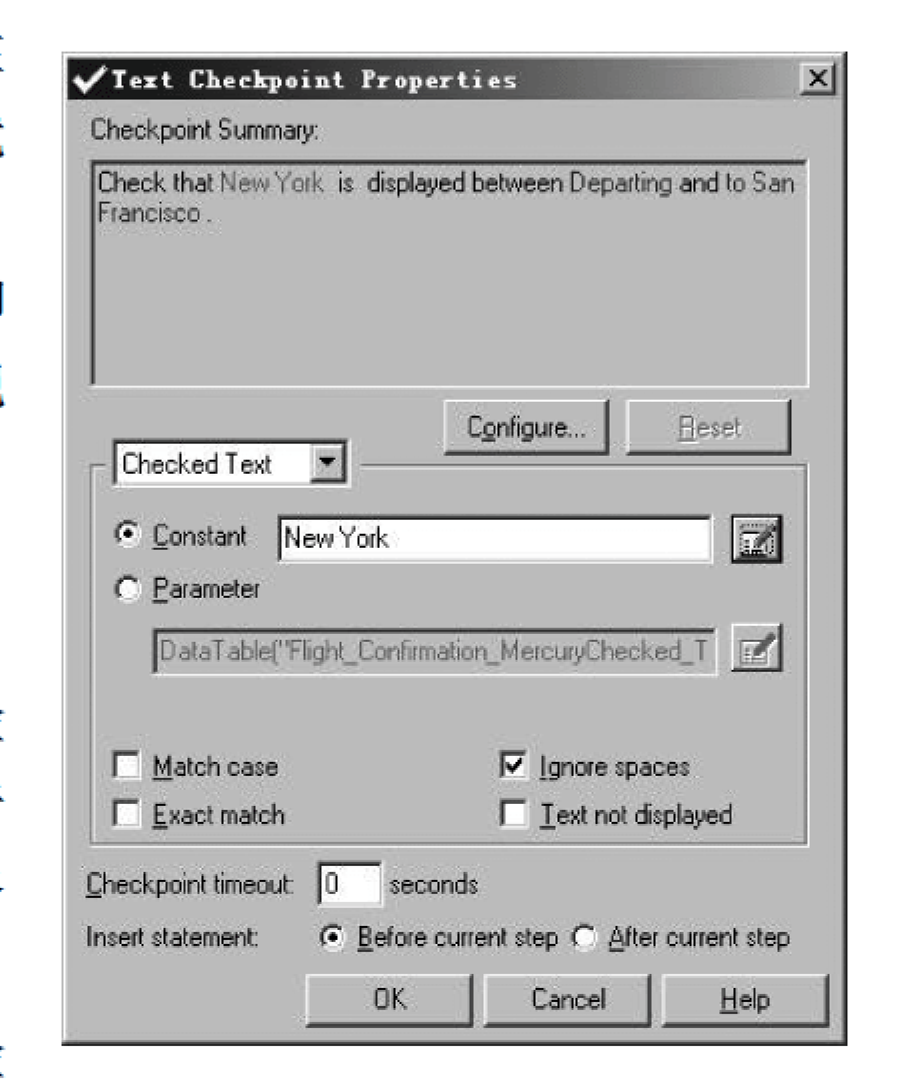


图 6.21　文本检查点属性设置

在 Test2 测试脚本中再添加一个表格检查点，检查 Book a Flight：Mercury 网页上航班的价格。创建表格检查点的具体步骤如下。

① 选取要建立检查点的网页。展开 Action1→Welcome：Mercury Tours 目录，选择 Book a Flight：Mercury 选项，在 Active Screen 窗口会显示相应的页面。

② 建立表格检查点。在 Active Screen 窗口中，在第一个航班的价钱“270”上右击，选择 Insert Standard Checkpoint 命令，打开 Object Selection-Checkpoint Properties 对话框，如图 6.22 所示。

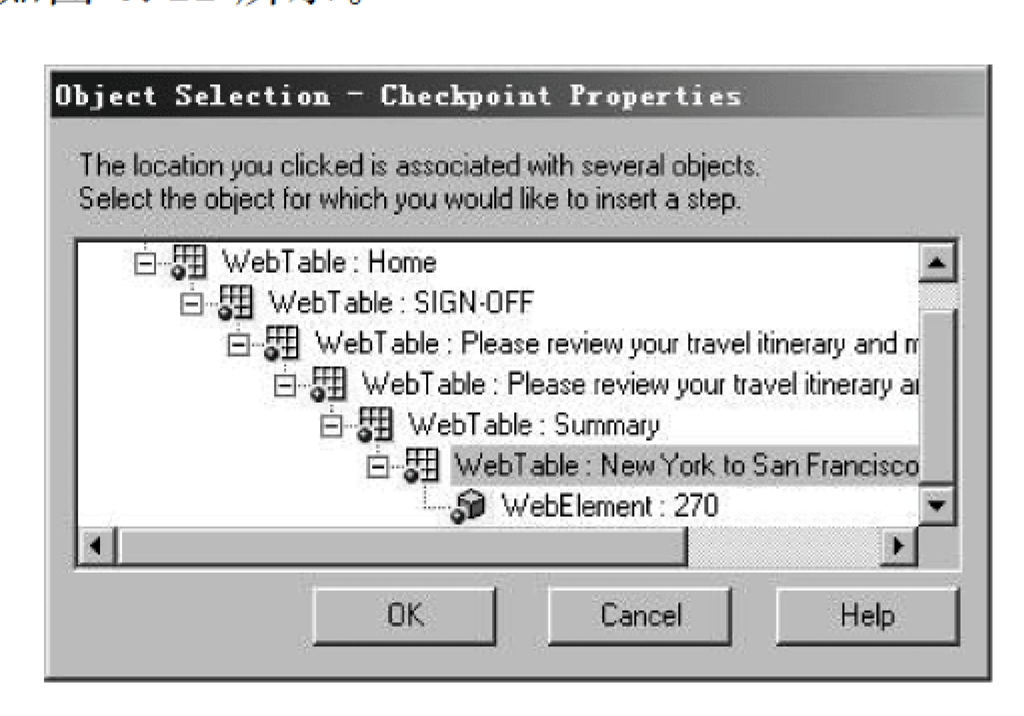


图 6.22　脚本对象属性图

刚打开时选取的是 WebElement:270，这时要选择上一层的 WebTable 对象，在这个例子中选择 WebTable：New York to San Francisco 选项。单击 OK 按钮打开 Table Checkpoint Properties 对话框，如图 6.23 所示，显示整个表格的内容。

预设每一个字段都会被选择，表示所有字段都会检查，可以对某个字段双击，取消检查字段，或者选择整个栏和列，执行选取或取消的动作。

在每个字段的列标题上双击，取消勾选的图标，然后在 270 字段处双击，这样执行时 QTP 只会对这个字段值作检查，如图 6.24 所示。

③ 单击 OK 按钮关闭对话框。QTP 会在测试脚本中 Book a Flight：Mercury 页面下加上一个表格检查点。

④ 在工具栏上单击 Save 按钮保存脚本。

(5) 执行并分析使用检查点的测试脚本

运行 Test2 测试脚本，分析插入检查点后脚本的运行情况。

在工具栏上单击 Run 按钮，弹出如图 6.25 所示对话框。

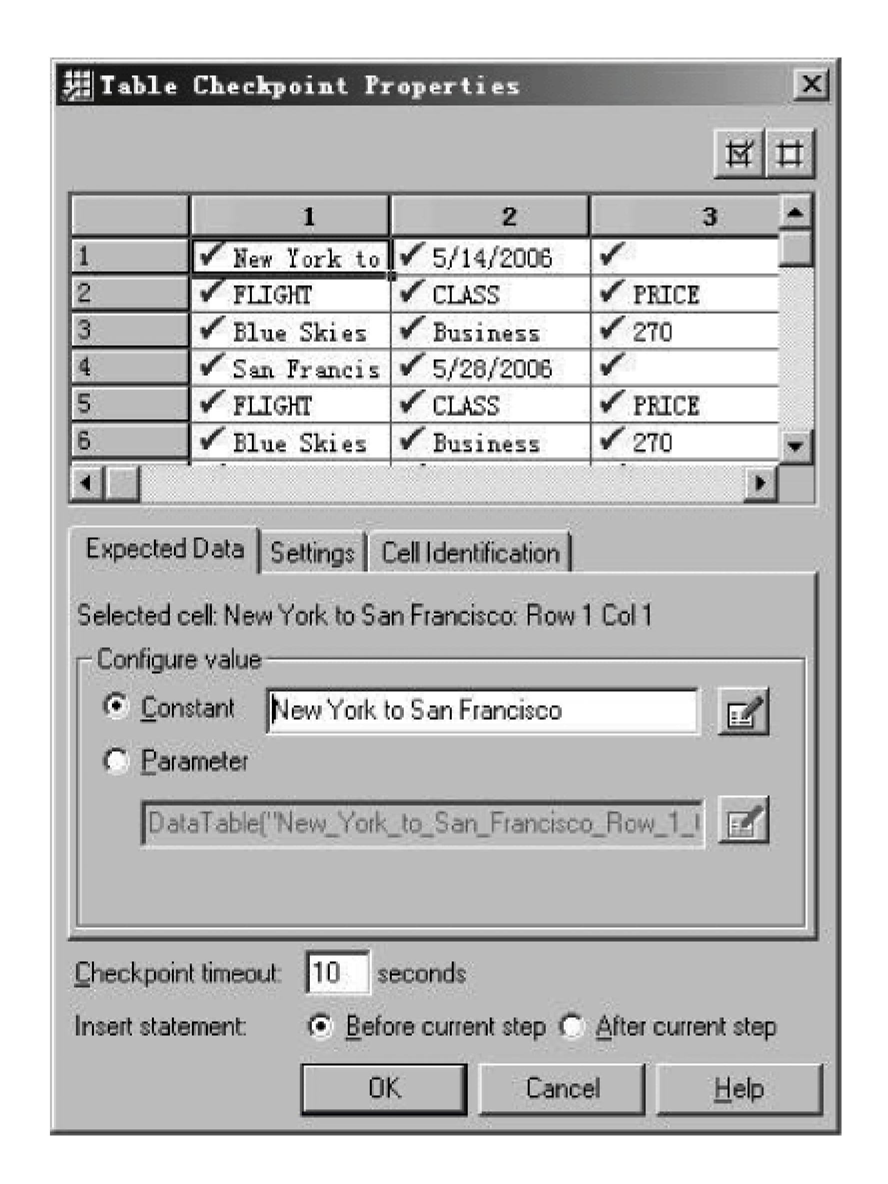


图 6.23　表格检查点属性设置

	1	2	3
1	New York to S	5/14/2006	
2	FLIGHT	CLASS	PRICE
3	Blue Skies Ai	Business	✓ 270
4	San Francisco	5/28/2006	
5	FLIGHT	CLASS	PRICE
6	Blue Skies Ai	Business	270

图 6.24　表格字段值检查设置

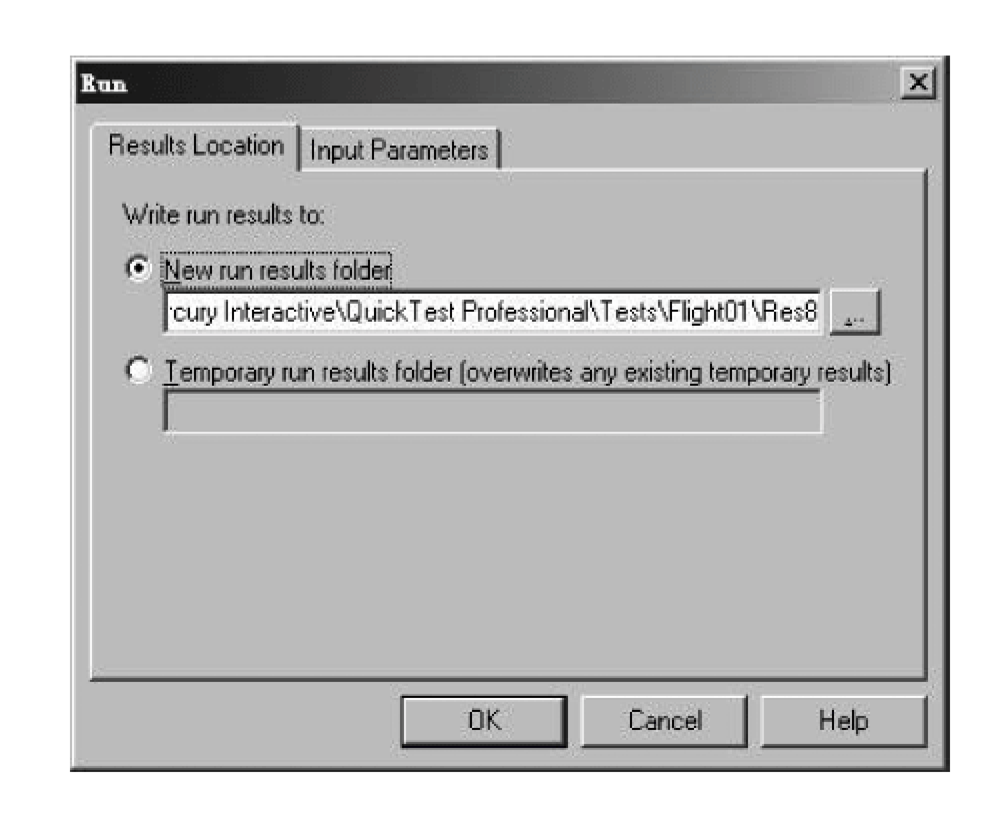


图 6.25　测试结果路径设置

这个页面是询问将本次测试结果保存在哪个目录，选中 New run results folder 单选按钮，接受默认设置，单击 OK 按钮确认。这时 QTP 会按照脚本中的操作，一步一步进行测试，操作过程和手工操作是完全一样的。

当 QTP 执行完测试脚本后，测试执行结果窗口会自动开启。如果所有的检查点都通过了验证，运行结果为 Passed。如果有一个或多个检查点没有通过验证，这行运行结果显示为 Failed，如图 6.26 所示。

在上图中可以看到，设置的 4 个检查点都通过了验证，下面看一下各个检查点的验证结果。

（1）验证网页检查点

在 Test Results 树视图中展开 Checkpoint Iteration 1 (Row 1)→Action1 Summary→Welcome: Mercury Tours→Book a Flight: Mercury 目录，并选择 Checkpoint“Book a Flight: Mercury”选项。

在右边的 Details 窗口中，可以看到网页检查点的详细信息。例如，网页检查点检查了哪些项目，如图 6.27 所示。

由于所有网页检查的项目，其实际值与预期值相符，所以这个网页检查点的结果为 Passed。

（2）验证表格检查点

在 Test Results 树视图中展开 Book a Flight: Mercury→New York to San Francisco 目录，并选择 Checkpoint“New York to San Francisco”选项。

在 Details 窗口可以看到表格的详细结果。也可以在下方看到整个表格的内容，被检查的字段以黑色的粗体文字显示，没有检查的字段以灰色文字显示，如图 6.28 所示。

这个表格检查点检查的字段值，其实际值与预期值相符，所以检查点的结果为 Passed。

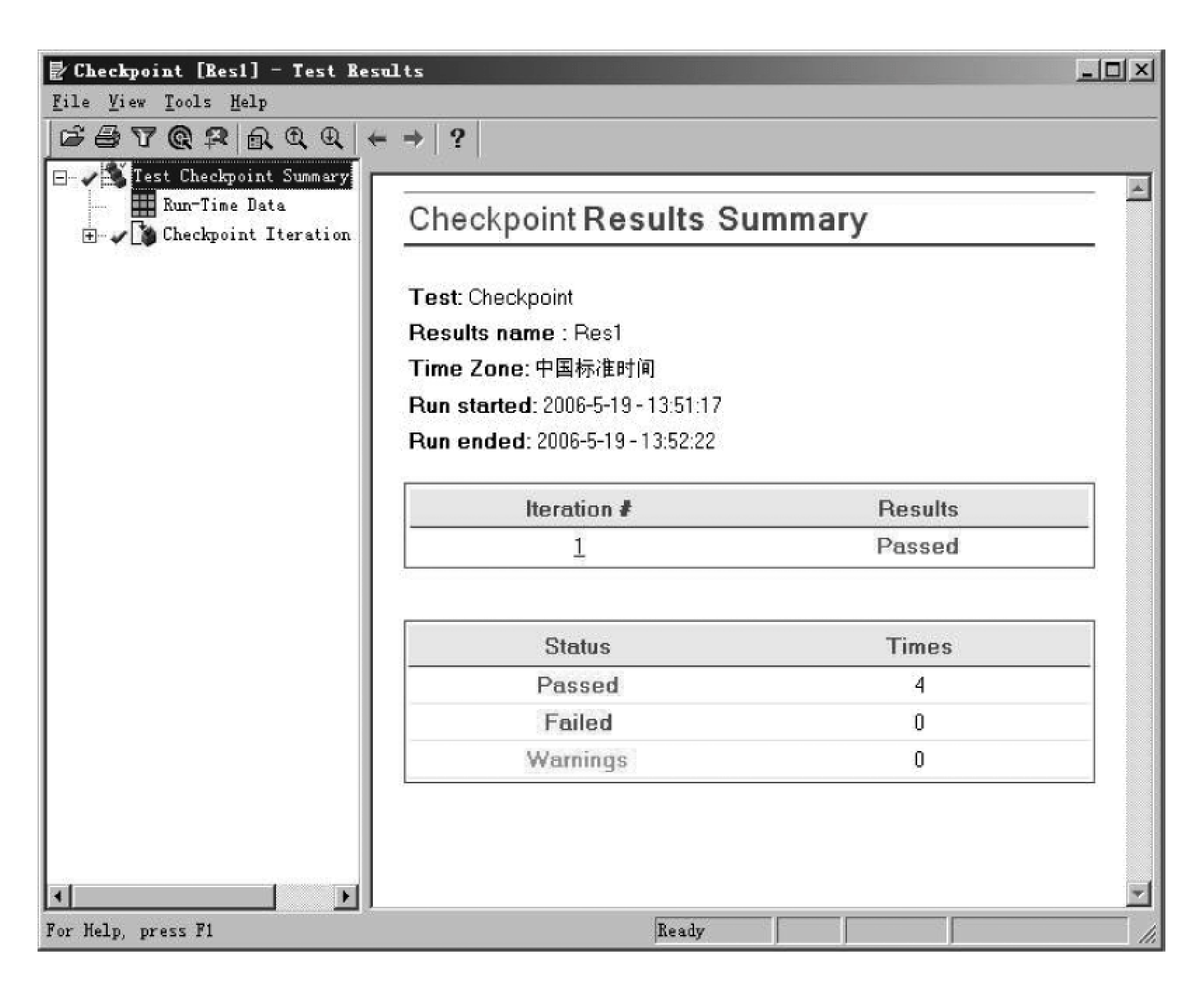


图 6.26　检查点测试结果

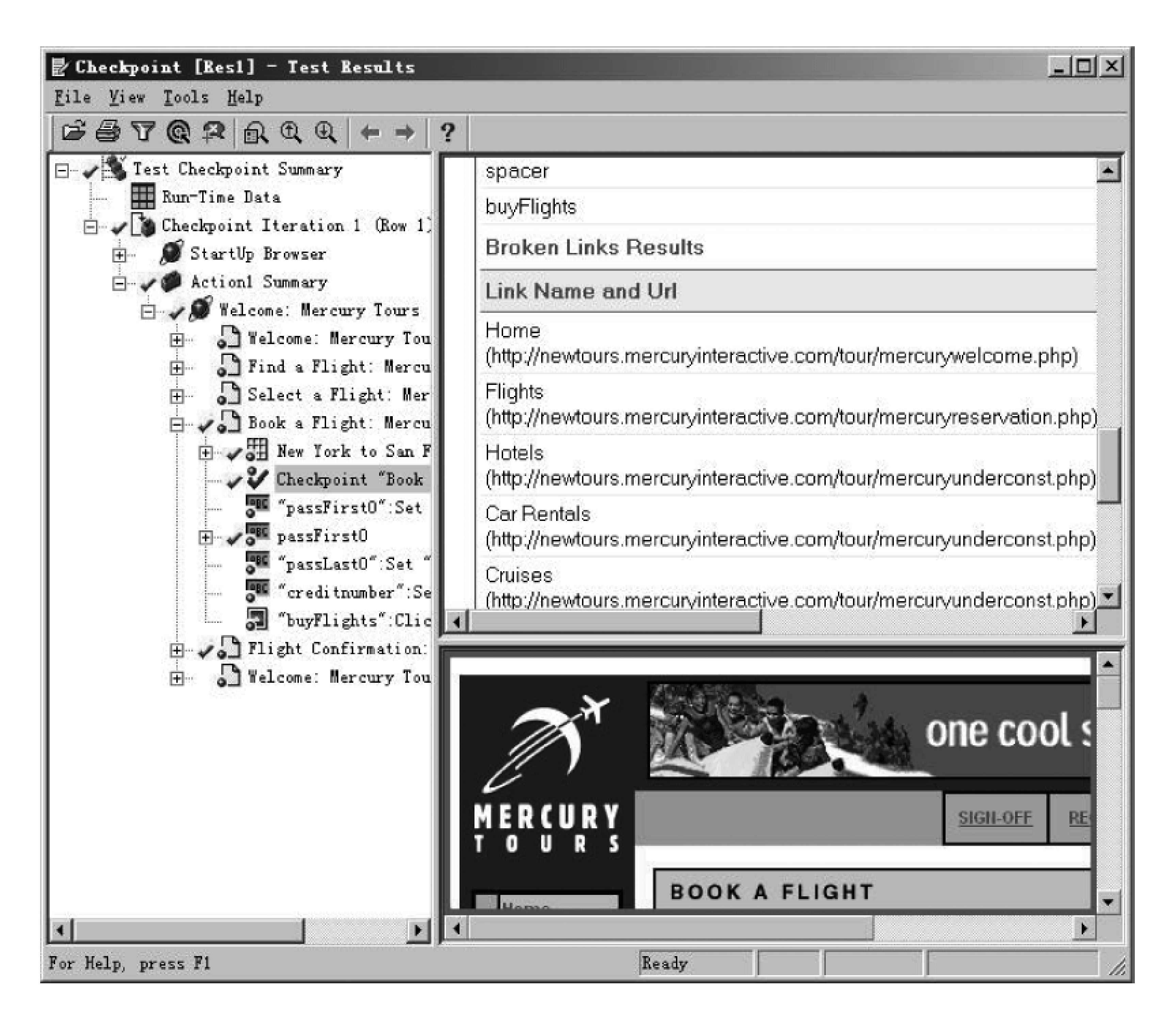


图 6.27　验证网页检查点

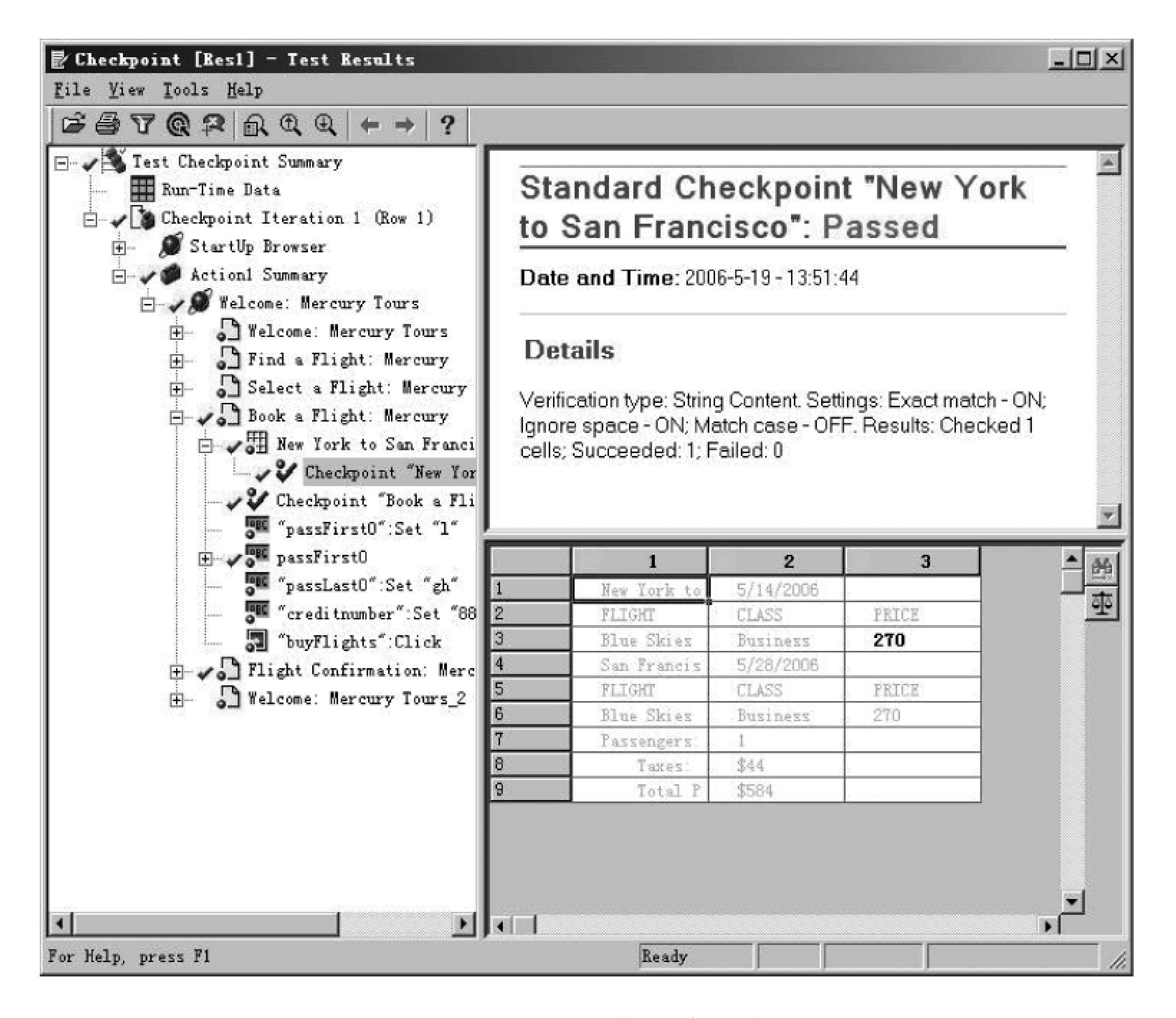


图 6.28　验证表格检查点

(3) 验证标准检查点

在 Test Results 树视图中展开 Book a Flight：Mercury→passFirst0 目录，并选择 Checkpoint“passFirst0”选项。在 Details 窗口可以看到标准检查点的详细结果，如检查了哪些属性以及属性的值，如图 6.29 所示。

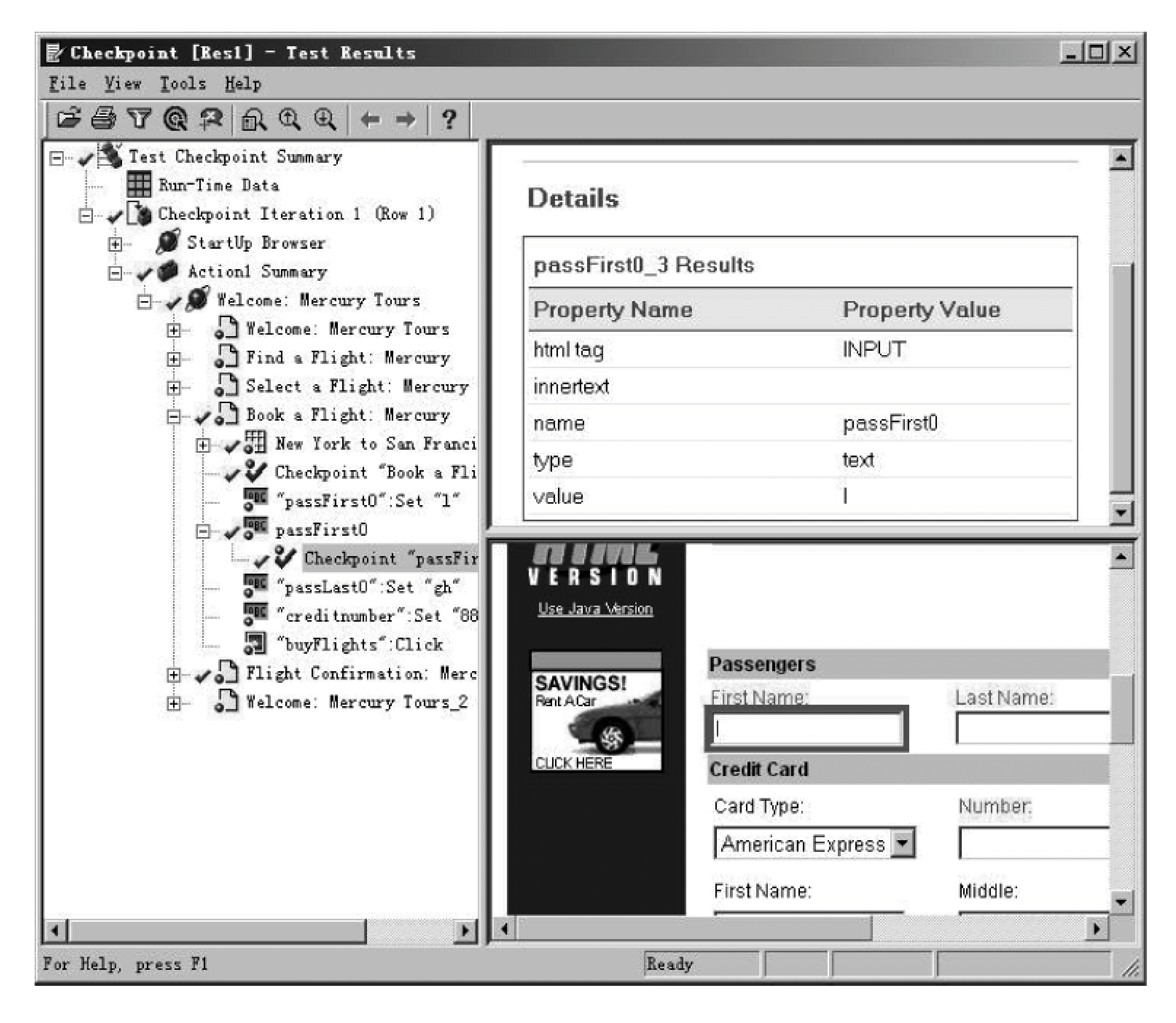


图 6.29　验证标准检查点

（4）验证文字检查点

在 Test Results 树视图中展开 Checkpoint Iteration 1（Row 1）→Action1 Summary→Welcome：Mercury Tours→Flight Confirmation：Mercury 目录，并选择 Checkpoint"New York"选项。显示如图 6.30 所示界面，因为文字检查点的实际值与预期值相同，所以检查点的结果为 Passed。

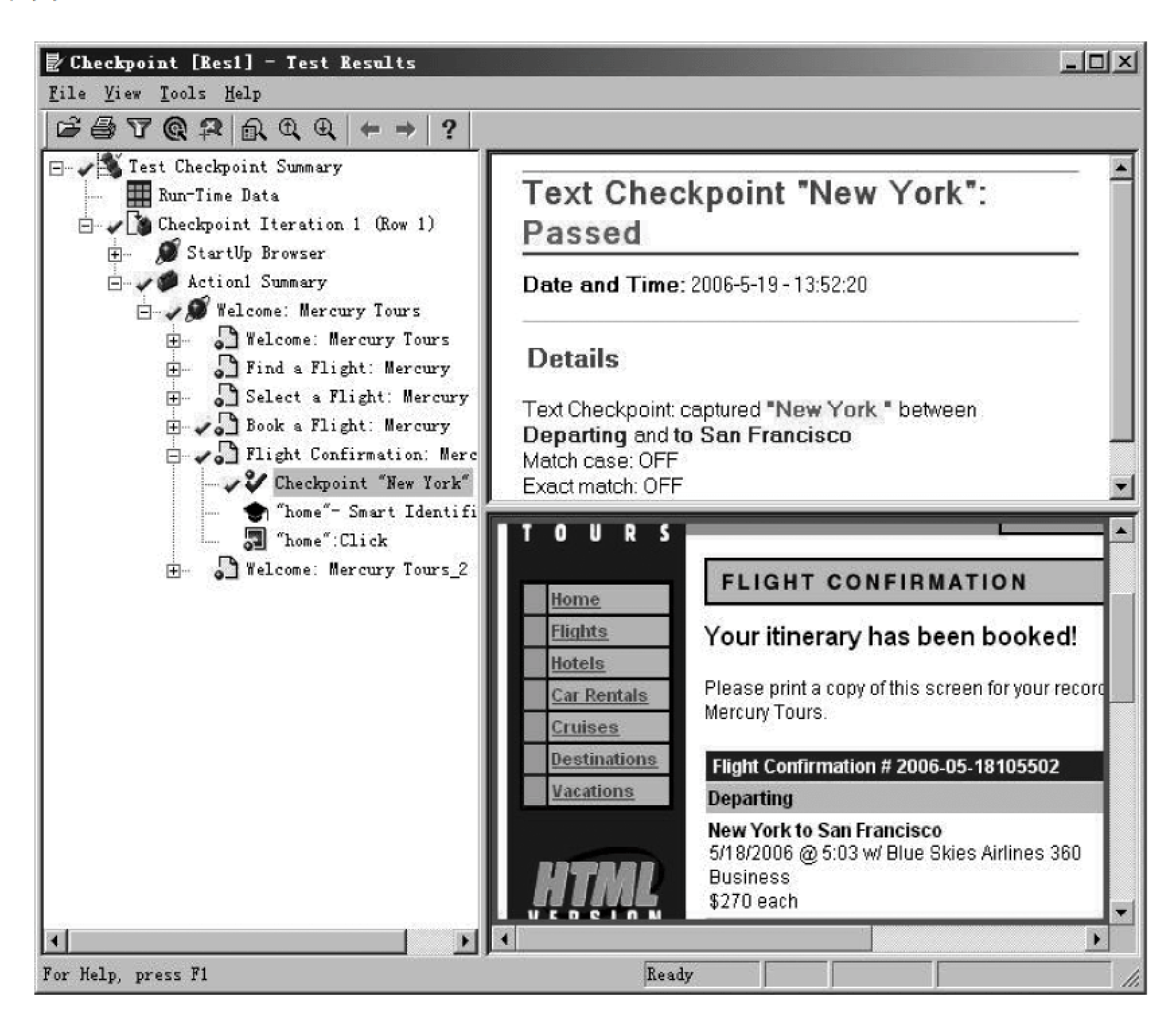


图 6.30　验证文字检查点

2. 参数化设置

在测试应用程序时，测试人员通常想检查对应用程序使用不同输入数据进行同一操作时，程序是否能正常地工作。在这种情况下，测试人员可以将这个操作重复录制多次，每次填入不同的数据，这种方法虽然能够解决问题，但实现起来比较笨拙。QTP 提供了一个更好的方法来解决这个问题，即对测试脚本参数化。参数化测试脚本包括数据输入的参数化和检测点的参数化。

使用 QTP 可以通过将固定值替换为参数，扩展基本测试或组件的范围。该参数化过程可以大大提高测试或组件的功能以及灵活性。

测试人员在 QTP 中使用参数功能，通过参数化测试或组件所使用的值来增强测试或组件。参数是一种从外部数据源或生成器赋值的变量。

QTP 可以参数化测试或组件中的步骤和检查点中的值，还可以参数化操作参数的值。如果希望参数化测试或组件多个步骤中的同一个值，可以考虑使用数据驱动器，而不是手动添加参数。

1）参数化步骤和检查点中的值

录制或编辑测试或组件时，可以参数化步骤和检查点中的值。可以参数化选定步骤的对象属性的值，还可以参数化为该步骤定义的操作（方法或函数参数）的值。

例如，应用程序或网站中可能包含一个带有编辑字段的表单，用户可以在该编辑字段中键入用户名，测试人员可能希望测试应用程序或网站是否读取该信息并将其正确显示在对话框中，这时候可以插入一个对已登录的用户名使用内置环境变量的文本检查点，以检查显示的信息是否正确。

通过参数化检查点属性的值，可以检查应用程序或网站如何基于不同的数据执行相同的操作。

例如，如果要测试 Mercury Tours 示例网站，可以创建一个检查点，以便检查预订机票后该机票是否被正确预订。假设测试人员需要检查针对各种不同目的地所预订的航班是否正确，可以为目的地信息添加一个数据表参数，而不是为每个目的地分别创建带有单独检查点的测试或组件。对于测试或组件的每次循环，QTP 都会针对不同目的地检查航班信息。

2）参数化对象和检查点的属性值

可以在“对象属性”或“对象库”对话框中参数化对象的一个或多个属性的值，也可以在“检查点属性”对话框中参数化检查点的一个或多个属性的值。

采用下列方式可以打开“对象属性”对话框或“检查点属性”对话框。

- 选择“步骤”→“对象属性”命令，或者右击某个步骤并选择“对象属性”命令，将打开“对象属性”对话框。
- 选择“工具”→“对象库”命令，单击“对象库”工具栏按钮，或者右击包含该对象的操作或组件，然后选择“对象库”命令，将打开“对象库”对话框。
- 选择“步骤”→“检查点属性”命令，或者右击该检查点并选择“检查点属性”命令，然后在对话框的“配置值”选项组中选择参数，如图 6.31 所示。

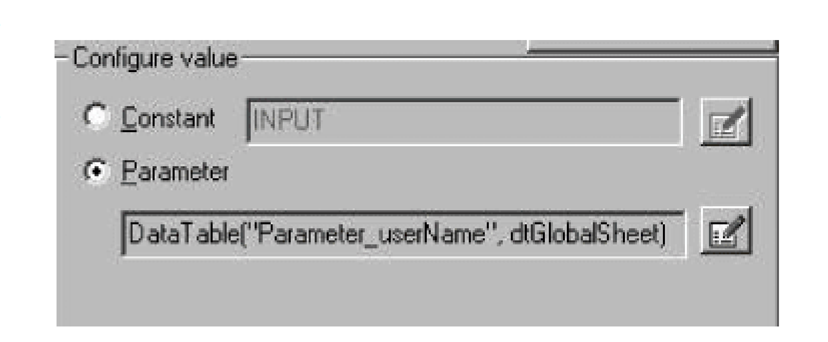


图 6.31　配置值

如果该值已经参数化，则“参数”框将显示该值的当前参数定义；如果该值尚未参数化，则“参数”框将显示该值的默认参数定义。

3）参数化操作的值

如果步骤中使用的方法或函数具有参数，则可以根据需要，参数化该参数值。例如，如果操作使用 Click 方法，则可以参数化 x 参数、y 参数或者这两者的值。

在关键字视图中选择已参数化的值时，将显示该参数类型的图标。例如，在以下片段中，已将 Set 方法的值定义为随机数字参数。每次运行测试或组件时，QTP 都会在 creditnumber 编辑框中输入一个随机数字值，如图 6.32 所示。



图 6.32　随机数字参数

可以使用视图的“值”列中的参数化图标来参数化操作值。

单击参数化图标，打开“值配置选项”对话框，将显示当前定义的值，如图 6.33 所示。

选择“参数”。如果该值已经参数化，则“参数”部分将显示该值的当前参数定义。如果该值尚未参数化，则“参数”部分将显示该值的默认参数定义。单击“确定”按钮，接受显示的参数语句并关闭该对话框。

选择一个尚未参数化的值时，QTP 会为该值生成默认参数定义。表 6.4 描述了如何确定默认参数设置。

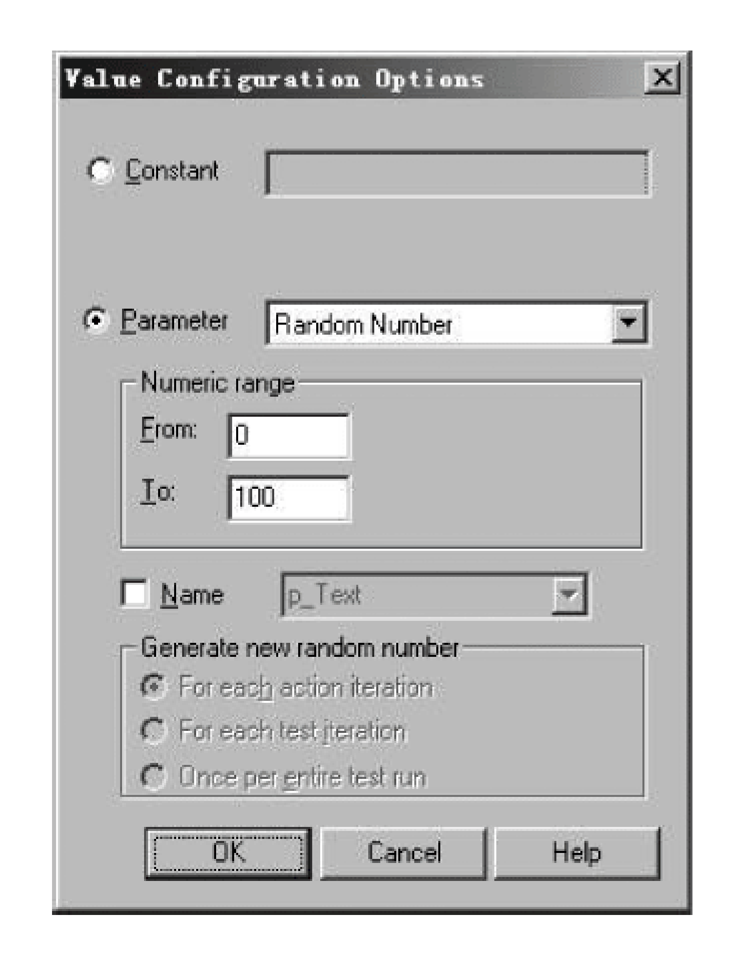


图 6.33 值配置选项

表 6.4 确定默认参数设置

执行参数化时	条 件	默认参数类型	默认参数名
操作中的步骤或检查点的值	至少在当前操作中定义了一个输入操作参数	操作参数	在“操作属性”对话框的“参数”选项卡中显示第一个输入参数
嵌套操作的输入操作参数值	至少为调用该嵌套操作的操作定义了一个输入操作参数	操作参数	在调用操作的“操作属性”对话框的“参数”选项卡中显示第一个输入参数
顶层操作调用的输入操作参数值	至少为测试定义了一个输入参数	测试参数	在“测试设置”对话框的“参数”选项卡中显示第一个输入参数
组件中的步骤或检查点的值	至少为该组件定义了一个输入参数	组件参数	在“业务组件设置”对话框的“参数”选项卡中显示第一个输入参数

如果上述相关条件不为真，则默认参数类型为“数据表”。如果接受了默认参数详细信息，QTP 将用基于选定值的名称新建一个数据表参数。

4）参数种类

QTP 有四种类型的参数，分别说明如下。

（1）测试、操作或组件参数

通过该类型参数可以使用从测试或组件中传递的值，或者来自测试中的其他操作的值。为了在特定操作内使用某个值，必须将该值通过测试的操作层次结构向下传递到所需的操作。然后，可以使用该参数值来参数化测试或组件中的步骤。

例如，假设要使用从运行（调用）测试的外部应用程序传递到测试中的某个值来参数化 Action3 中的一个步骤，可将该值从测试级别传递到 Action1（顶层操作）至 Action3（Action1 的子操作），然后使用该“操作”输入参数值（从外部应用程序传递的值）来参数化所需的步骤。

（2）数据表参数

通过该类型参数可以让测试人员所提供的数据多次运行。在每次重复（或循环）中，QTP 均使用数据表中不同的值。例如，假设应用程序或网站包含一项功能，用户可以通过该功能从成员数据库中搜索联系信息。当用户输入某个成员的姓名时，将显示该成员的联系信息，以及一个标记为“查看＜MemberName＞的照片”的按钮，其中＜MemberName＞是该成员的姓名。可以参数化按钮的名称属性，以便在运行会话的每次循环期间，QTP 可标识不同的照片按钮。

（3）环境变量参数

通过它可以在运行会话期间使用来自其他来源的变量值。这些变量值可以是测试人员所提供的值，或者是 QTP 基于测试人员选择的条件和选项而生成的值。例如，可以让 QTP 从某个外部文件读取用于填写 Web 表单的所有值，或者可以使用 QTP 的内置环境变量之一来插入有关运行测试或组件的计算机的当前信息。

（4）随机数字参数

通过它可以插入随机数字作为测试或组件的值。例如，要检查应用程序处理大小机票订单的方式，可以让 QTP 生成一个随机数字，然后将其插入到“票数”编辑字段中。

5）使用参数

（1）使用数据表参数

可以通过创建数据表参数来为参数提供可能的值列表。通过数据表参数可以创建使用所提供的数据多次运行的数据驱动测试、组件或操作。在每次重复中，QTP 均使用数据表中不同的值。

例如，在 Mercury Tours 示例网站中，通过该网站可预订航班请求。要预订航班，需要提供航班路线，然后单击“继续”按钮。该网站将针对请求的路线返回可用的航班。

测试人员可以通过访问网站并录制大量查询的提交来执行该测试，但这是一个既费时又费力的低效解决方案。通过使用数据表参数，可以连续对多个查询运行测试或组件。

参数化测试或组件时，需要首先录制访问网站并针对所请求的一条路线来检查可用航班的步骤，然后将录制的路线替换为某个数据表参数，并在数据表的全局表中添加自己的数据集，每条路线一个，如图 6.34 所示。

A1 New York

	Departure	arrival	C	D	E	F	G
1	New York	San Francisco					
2	London	Paris					
3	Frankfure	Acapulco					
4							
5							
6							
7							
8							

Global Action1

图 6.34　使用数据表参数

新建数据表参数时，将在数据表中添加新的一列，并将参数化的当前值放在第一行中。如果要对值进行参数化并选择现有的数据表参数，则将保留所选参数的列中的值，并且这些值不会被参数的当前值覆盖。

表中的每个列都表示单个数据表参数的值列表，列标题是参数名。表中的每一行都表

示 QTP 在测试或组件的单次循环期间为所有参数提交的一组值。运行测试或组件时，QTP 将针对表中的每一行数据运行一次测试或组件。例如，如果测试在数据表的全局表中有十行，则运行十次循环。

(2) 使用环境变量参数

QTP 可以插入环境变量列表中的值，该列表是可通过测试访问的变量和相应值的列表。在测试运行的整个过程中，无论循环次数是多少，环境变量的值始终保持不变，除非在脚本中以编程方式更改变量的值。

QTP 有以下三种环境变量：用户定义的内部环境变量、用户定义的外部环境变量以及内置环境变量。

① 用户定义的内部环境变量是在测试内定义的变量。这些变量与测试一起保存，并且只能在定义这些变量的测试内访问。在“测试设置”对话框或“参数选项”对话框的“环境”选项卡中，可以创建或修改测试中用户定义的内部环境变量。

② 用户定义的外部环境变量是活动外部环境变量文件中预定义的变量。可根据需要创建任意多的文件，并为每个测试选择一个适当的文件，或者更改用于每个测试运行的文件。

③ 内置环境变量是表示有关测试和运行测试的计算机的信息的变量。例如测试路径和操作系统。从所有测试和组件中都可以访问这些变量，并且它们都被指定为只读变量。

(3) 使用随机数字参数

当选择“随机数字”作为参数类型时，可以通过“参数选项”对话框将参数配置为使用随机数字，如图 6.35 所示。“值配置选项”对话框的“参数”部分与“参数选项”对话框非常相似。

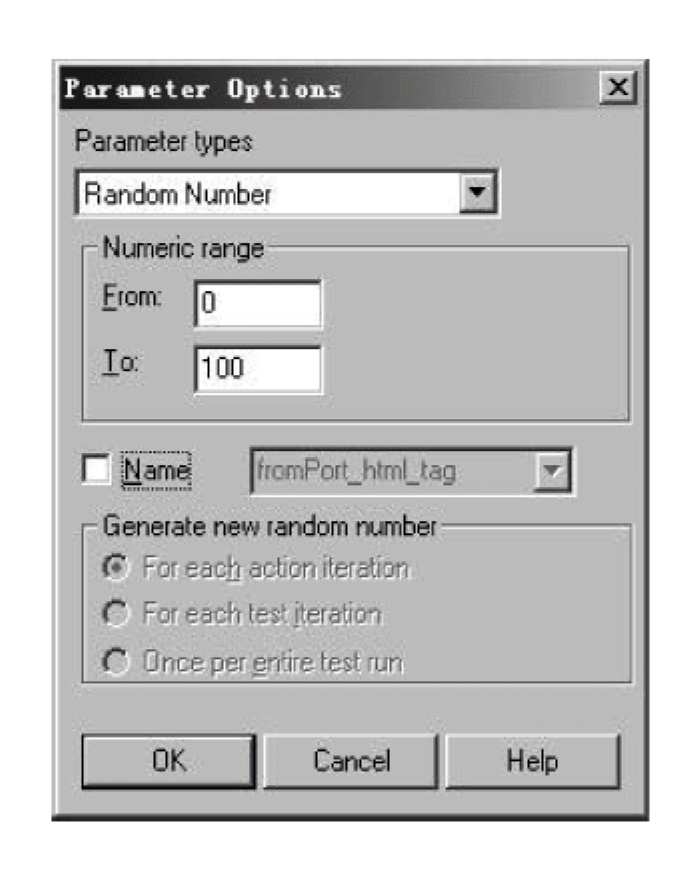


图 6.35 使用随机数字参数

数字范围是指定用于生成随机数字的范围。默认情况下，随机数字范围介于 0～100 之间。可通过在“从”和“到”框中输入不同的值来修改此范围。该范围必须介于 0～2147483647(包含)之间。

名称是指定参数的名称。通过为随机参数指定名称可以在测试中多次使用同一个参数。可以选择现有的命名参数，或者通过输入新的描述性名称来新建命名参数。

生成新随机数字。选中“名称”复选框时会启用该框。可以选择下列选项之一：为每次操作循环(在每次操作循环结束时生成一个新数字)；为每次测试循环(在每次全局循环结束时生成一个新数字)；为整个测试运行生成一次(第一次使用参数时生成一个新数字。在整个测试运行中，对参数使用同一个数字)。

6) 参数化测试脚本

在测试脚本 Test2 中，“纽约”是个常数值，每次执行测试脚本预订机票时，出发地点都是纽约。现在，我们将测试脚本中的出发地点参数化，这样，执行测试脚本时就会以不同的出发地点去预订机票。

首先，打开 Test2 测试脚本，将脚本另存为 TestParameter，然后选择要参数化的文字，具体如下。

在视图树中展开 Action1→Welcome：Mercury Tours→Find a Flight：Mercury 目录，选择 fromPort 右边的 Value 字段，然后单击参数化图标，开启 Value Configuration

Options 对话框，如图 6.36 所示。

设置要参数化的属性，选择 Parameter 选项，这样就可以用参数值来取代 New York 这个常数。在参数中选择 Data Table 选项，这样这个参数就可以从 QTP 的 Data Table 中取得，将参数的名字改为 departure，如图 6.37 所示。



图 6.36 “值配置选项”对话框

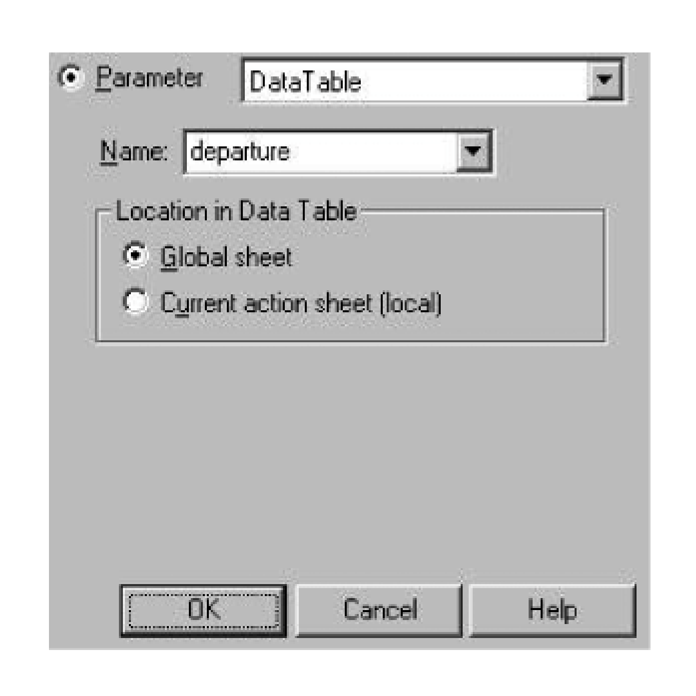


图 6.37 设置参数化的属性

单击 OK 按钮确认，QTP 会在 Data Table 中新增 departure 参数字段，并且插入了一行 New York 的值，New York 会成为测试脚本执行使用的第一个值，如图 6.38 所示。

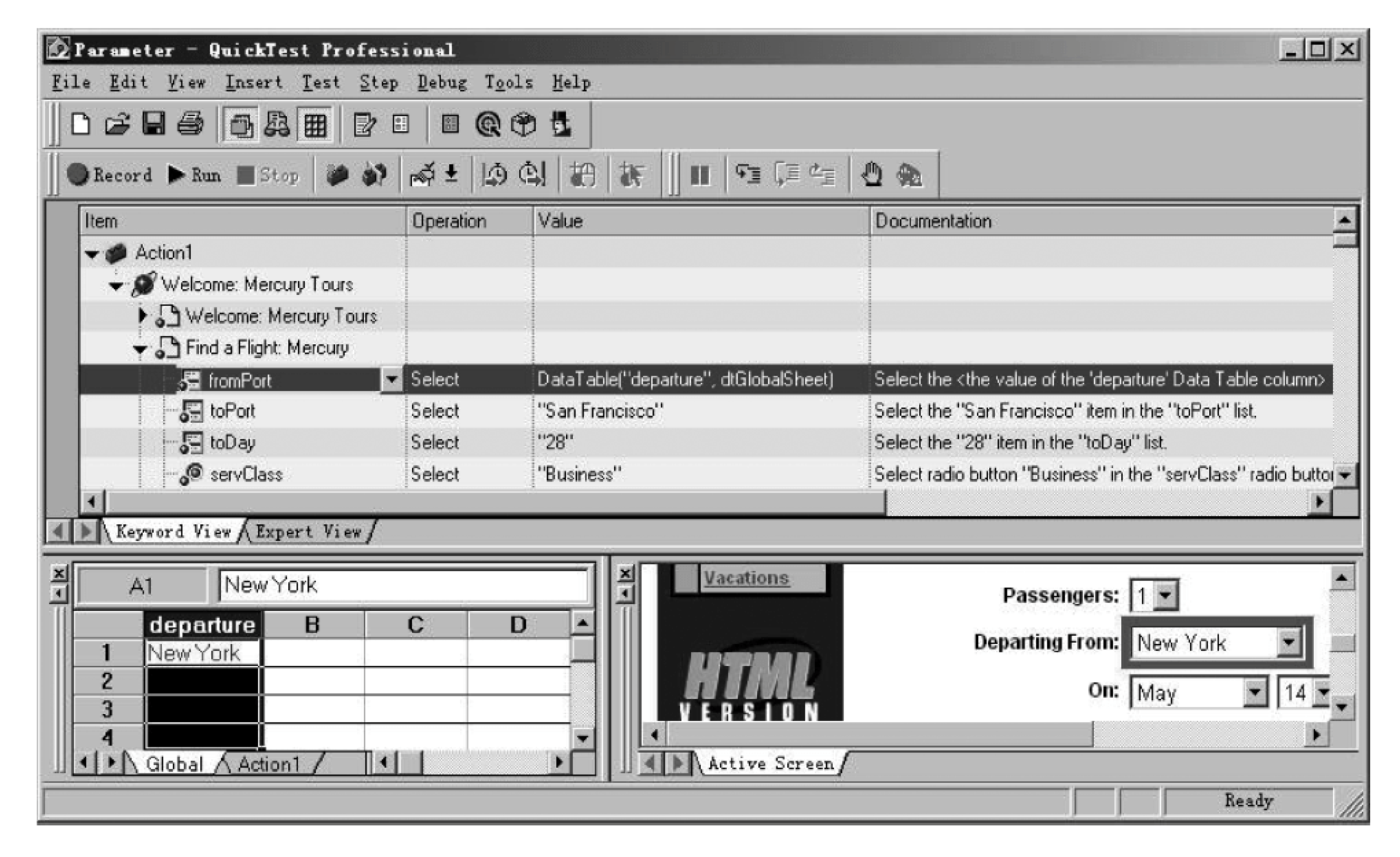


图 6.38 测试步骤参数化

参数化以后可以看到树视图中的变化，在参数之前，这个测试步骤显示“fromPost… Select… New York”，现在，这个步骤变成了“fromPost …Select… Data Table (“departure”, dtGlobalSheet)”。而且当单击 Value 字段时，Value 字段会显示如图所示：<departure>，表示此测试步骤已经被参数化，而且其值从 Data Table 的 departure 字段中获得。

在 departure 字段中加入出发点资料，使 QTP 可以使用这些资料执行脚本。在

departure 字段的第二行、第三行分别输入：Portland、Seattle。

保存测试脚本。

7）修正受到参数化影响的步骤

当测试步骤被参数化以后，有可能会影响到其他的测试步骤。例如，为了验证在 Flight Confirmation 网页中是否出现 New York，在网页上添加了一个文字检查点，那么，就要对出发地的文字检查点作参数化，以符合对出发地点参数化的预期结果。

修正文字检查点。首先在树视图中，展开 Action1→Welcome：Mercury Tours→Flight Confirmation：Mercury 选项，然后在其上右击选择 Checkpoint Properties 命令，打开 Text Checkpoint Properties 对话框，如图 6.39 所示。

在 Checked Text 的 Constant 文本框中显示为 New York，表示测试脚本在每次执行时，这个文字检查点的预期值都为 New York。我们选择 Parameter，单击旁边的 Parameter Options 按钮，打开 Parameter Options 对话框，如图 6.40 所示。

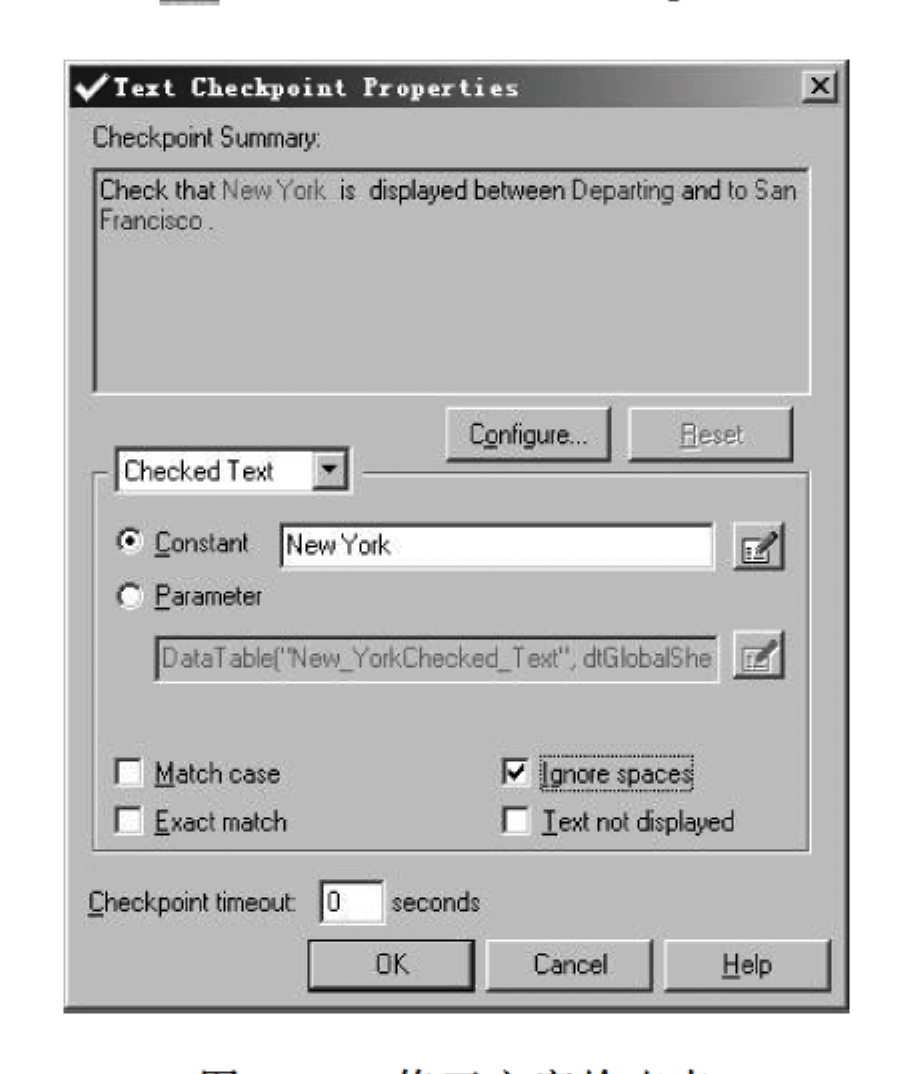


图 6.39　修正文字检查点

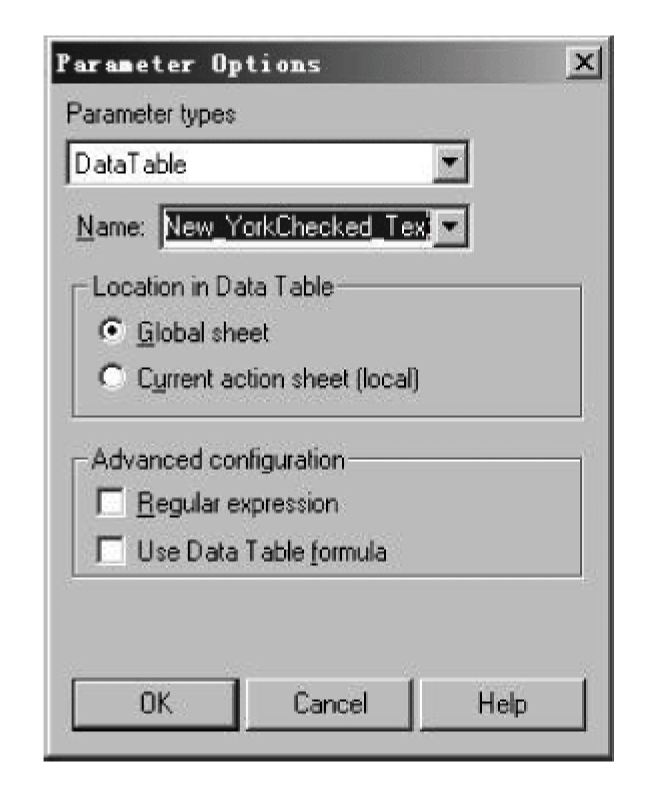


图 6.40　“参数选项”对话框

在“参数类型”下拉列表框中选择 Data Table 选项，在“名称”下拉列表框中选择 departure 选项，指明这个文字检查点使用 departure 字段中的值当成检查点的预期值。

单击 OK 按钮关闭窗口，这样文字检查点也被参数化了。

8）执行并分析使用参数的测试脚本

参数化测试脚本后，运行 TestParameter 测试脚本。QTP 会使用 Data Table 中的 departure 字段值，执行三次测试脚本。

执行测试脚本。单击工具栏上的 Run 按钮，开启 Run 对话框，选择 New run results folder 选项，其余为默认值，单击 OK 按钮开始执行脚本。当脚本运行结束后，会开启测试结果窗口。

在树视图中，展开 Parameter Iteration 2→Action1 Summary→Welcome Mercury Tours→Flight Confirmation：Mercury 目录，选择 Checkpoint“New York”选项，显示如图 6.41 所示。

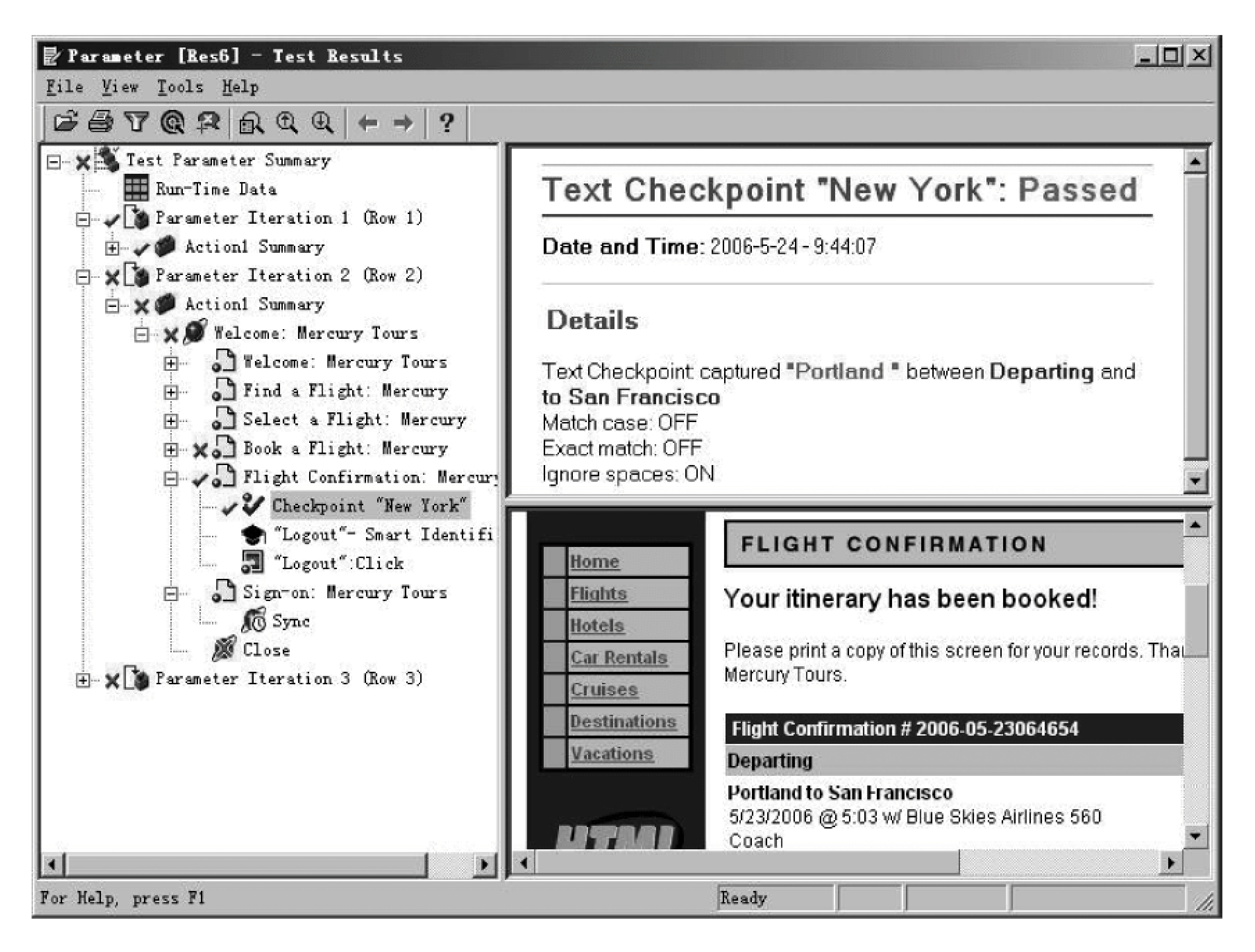


图 6.41　测试结果窗口

在检查点 Details 窗口中，显示 Portland 为预期结果同时也是实际的值，所以文字检查点为通过。同时也可以看到在下方的 Application 窗口中，显示机票的出发地点也是 Portland。

在图中可以看出，虽然每次执行时，文字检查点的结果是通过的，但是第二次与第三次的执行结果仍然为失败。这是因为出发地点的改变，造成在表格检查点中的机票价钱改变，导致表格检查点失败。在以后的课程中，将学习修正表格检查点，让 QTP 自动更新表格检查点的预期结果，就可以检查正确的票价。

3. 输出值

通过 QTP 可以检索测试或组件中的值，并将这些值作为输出值存储。此后，就可以检索这些值，并在运行会话的不同阶段使用该值作为输入。

输出值是一个步骤，在该步骤中，捕获测试或组件中某个特定点的一个或多个值，并在运行会话持续时间存储这些值。随后，在运行会话中的不同点，可以将这些值作为输入使用。

可以输出任何对象的属性值。还可以从文本字符串、表单元格、数据库和 XML 文档输出值。

创建输出值步骤时，可以确定运行会话持续时间内的值存储在哪里，以及如何使用这些值。运行会话期间，QTP 检索指定点的每个值并将其存储在指定位置，以后当运行会话中需要值时，QTP 将从该位置检索值并根据需要来使用。

1）输出值类型

将输出值步骤添加到测试或组件时，首先选择要输出的值的类别，例如，属性值、文本值或 XML 元素值；然后，就可以确定要输出的值以及每个值的存储位置。在 QTP 中可以

创建以下几个类别的输出值：标准输出值、文本/文本区输出值、数据库输出值、XML 输出值。

(1) 标准输出值

可以使用标准输出值来输出大多数对象的属性值。例如，在基于 Web 的应用程序中，一个网页中的链接数可能基于用户在上一页的表单中所做选择的不同而变化。可以在测试中创建一个输出值，来存储页面中的链接数，也可以使用标准输出值来输出表单元格的内容。

(2) 文本/文本区输出值

可以使用文本输出值来输出屏幕或网页中显示的文本字符串。创建文本输出值时，可以输出对象文本的一部分，也可以指定要在输出文本之前和之后输出的文本。

可以使用文本区域输出值来输出 Windows Applications 中屏幕已定义区域内显示的文本字符串。例如，假设在测试的应用程序中，想要存储显示在特定步骤之后的任何错误消息的文本。在 If 语句中，查看带有已知标题栏值(例如 Error)的窗口是否存在。如果该窗口存在，则输出该窗口中的文本(假设窗口大小与所有可能的错误消息的大小相同)。

在基于 Windows 的应用程序中创建文本或文本区输出值时使用文本识别机制，有时会检索到不想要的文本信息(例如隐藏文本和带阴影的文本，这些文本会作为同一字符串的多个副本显示)。

此外，在不同的运行会话中，文本(和文本区)输出值的表现方式可能不同，具体取决于使用的操作系统版本、已经安装的 Service Pack、安装的其他工具包、应用程序中使用的 API 等。

(3) 数据库输出值

可以使用数据库输出值，基于在数据库上定义的查询的结果(结果集)来输出数据库单元格内容的值。可以从结果集的全部内容中创建输出值，也可以从其中某一部分创建输出值。在运行会话过程中，QTP 从数据库中检索当前数据，并根据指定的设置来输出值。

(4) XML 输出值

可以使用 XML 输出值输出 XML 文档中的 XML 元素和属性的值。运行会话完成后，可以在“测试结果”窗口中查看 XML 输出值的概要结果，也可以通过打开“XML 输出值结果”窗口来查看详细结果。例如，假设网页中的某个 XML 文档包含新车的价目表，可以通过选择要输出的相应的 XML 元素值来输出特定汽车的价格。

表 6.5 给出每种环境支持的输出值类型。

2) 存储输出值

定义输出值时，可以指定运行会话期间在哪里以及如何存储每个值。可以将值输出到测试、操作或组件参数，运行时数据表，环境变量中。

(1) 将值存储在测试、操作或组件参数中

可以将值输出到操作或组件参数，以便可以在运行会话后面的部分中使用来自运行会话某一部分的值，或者传递回运行(调用)测试或组件的应用程序。

表 6.5　每种环境支持的输出值类型

输出值类别	Web	Windows	VB	ActiveX	其 他 环 境
标准	S	S	S	S	NA
页(标准)	S	NA	NA	NA	NA
表(标准)	S	NA	NA	S	NA
文本	S	S	S	S	NA
文本区	NS	S	S	S	NA
数据库	NS	NA	NA	NA	S(DbTable)
XML	S	NA	NA	NA	XML 文件

＊S—支持　NS—不支持　NA—不适用

例如,假设要测试一个购物应用程序,该程序计算采购费用,并自动从账户中扣除采购金额。想要测试在每次运行带有不同的采购单的操作或组件时,该应用程序是否能够正确地从账户中扣除采购金额,可以将花费的总金额输出到某个操作或组件的参数值,然后在稍后的扣除该金额操作中的运行会话部分使用该值。

(2) 将值存储在运行时数据表中

对于要运行多次的由数据驱动的测试(或操作)来说,将值输出到运行时数据表的选项十分有用。在每次重复或循环中,QTP 检索当前值并将其存储在运行时数据表的相应的行中。

例如,要测试一个航班预订应用程序,因此设计了一个测试来创建新预定,随后查看预定详细信息。每次运行测试时,应用程序为新预定生成一个唯一的订单号。要查看预订,应用程序要求用户输入相同的订单号。可是在运行该测试之前,还不知道订单号。

要想解决这个问题,就要将在创建新预定时生成的唯一订单号的值输入数据表中。然后,在 View Reservation 屏幕中,使用包含存储值的列将输出值插入订单号输入字段中。运行测试时,QTP 检索站点为新预定生成的唯一订单号,并在运行时数据表中输入此输出值。测试到达查看预订所需的订单号输入字段时,QTP 将存储在运行时数据表中的唯一订单号插入订单号字段中。

(3) 将值存储在环境变量中

将值输出到内部用户定义的环境变量时,可以在运行会话后面的阶段使用该环境变量输入参数。

例如,假设在测试一个应用程序,该程序会提示用户在“欢迎使用”页输入账号,然后显示用户姓名。就可以使用文本输出值来捕获显示的名称值,并将其存储在环境变量中。然后,可以检索环境变量中的值以便在应用程序的其他位置中输入用户的姓名。

3) 输出属性值

(1) 定义标准输出值

通过“输出值属性”对话框可以选择要输出的属性值,并定义所选择的每个值的设置,如图 6.42 所示。

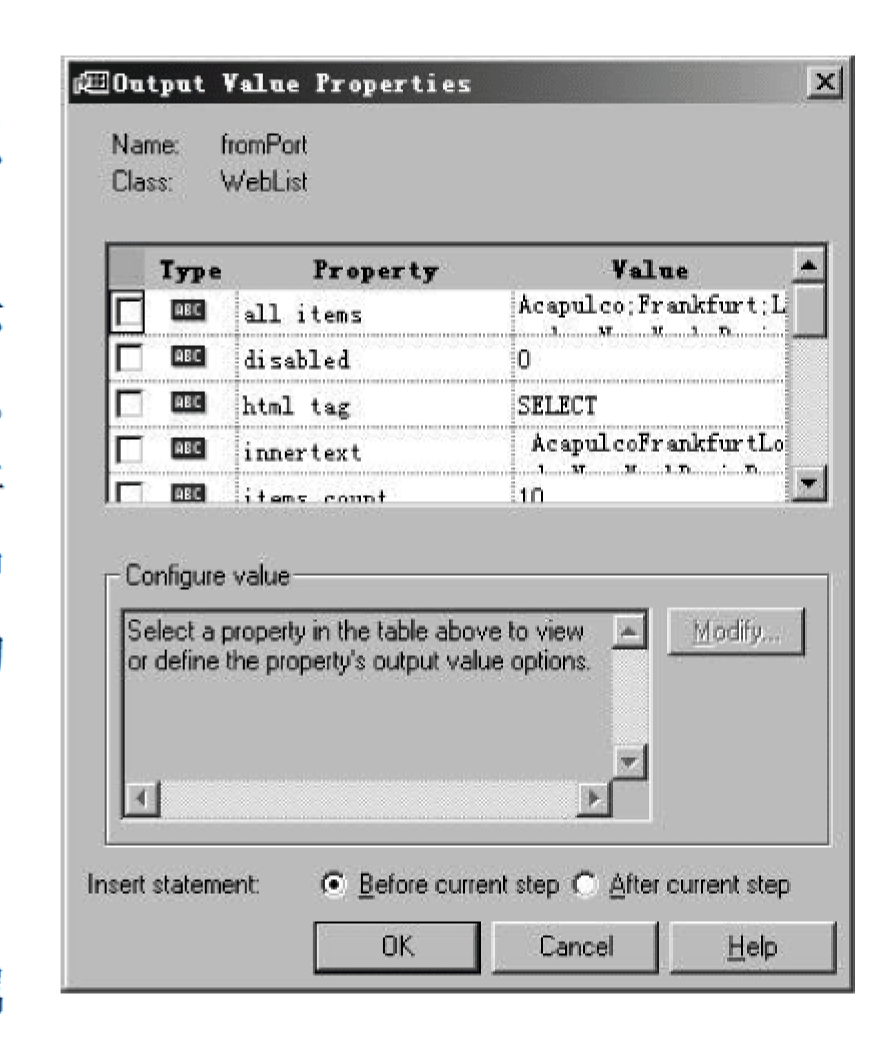


图 6.42　输出值属性

关闭此对话框之前，可以为相同对象选择许多属性并为每个属性值定义输出设置。运行会话过程中到达输出值步骤时，QTP 将检索所有指定的属性值。

① 标识对象。对话框的上部显示有关要创建输出值的测试对象的信息，如表 6.6 所示。

表 6.6　标示对象信息

项目	描　　述
名称	测试对象的名称
类	对象的类别

② 选择要输出的属性值。对话框的上半部分包含一个窗格，其中列出选定对象的属性，以及它们的值和类型。该窗格包含项如表 6.7 所示。

表 6.7　对象的属性

窗口元素	描　　述
复选框	要指定将输出的属性，则选择相应的复选框，可以为对象选择多个属性，并为选择的每个属性值指定输出选项
类别	ABC 图标表示属性的值当前为常量； 图标表示属性的值当前存储在测试、操作或组件参数中； 图标表示属性的值当前存储在运行时数据表中； 图标表示属性的值当前存储在环境变量中
属性	属性的名称
值	属性的当前值

③ 指定属性值的输出设置。选择属性的复选框时，将突出显示属性详细信息，并且在“配置值”区域中显示选定属性值的当前输出定义，如图 6.43 所示。

第一次选择要输出的属性值时，“配置值”区域中会显示值的默认输出定义。选择要输出的属性值时，可以通过选择其他属性值或单击 OK 按钮接受显示的输出定义；通过单击“修改”按钮更改选定值的输出类型和设置。

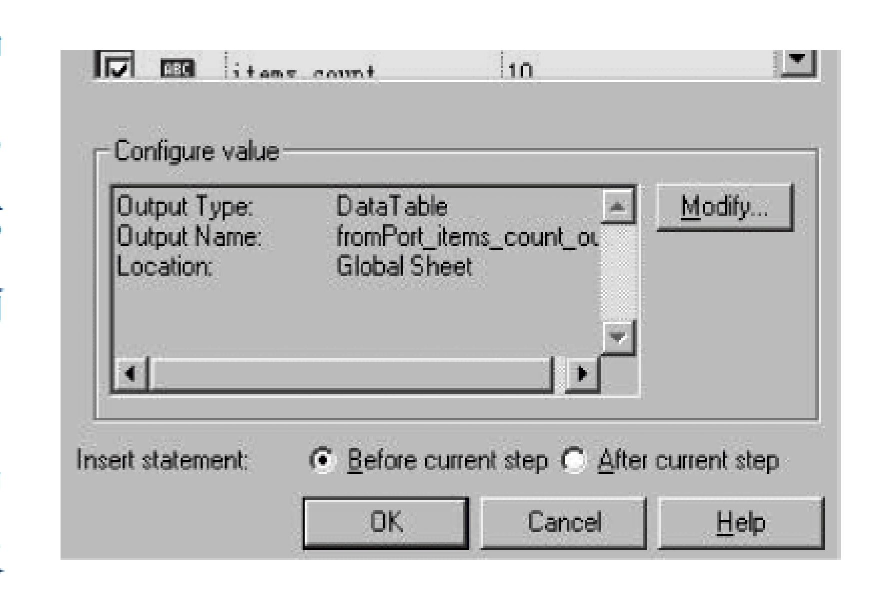


图 6.43　选定属性值的当前输出定义

④ 指定输出类型和设置。为每个值定义的输出类型和设置决定该值在运行会话中的存储位置以及使用方式。到达输出值步骤时，QTP 检索为输出选定的每个值并将其存储在指定位置，以供以后在运行会话中使用。新建输出值步骤时，QTP 为选定要输出的每个值指定一个默认定义。

可以通过选择不同的输出类型或更改输出设置来更改选定值的当前输出定义：将值输出到操作或组件参数；将值输出到数据表；将值输出到环境变量。

(2) 将值输出到操作或组件参数

可以将值输出到操作或组件参数,以便这些值可以在运行会话的后面部分中使用,或者传递回运行(调用)测试或组件的外部应用程序。如果参数已经定义为用于调用操作或组件的输出参数,则只能将值输出到操作或组件参数。此外,仅当输出值类型和参数值类型匹配时,将值输出到操作或组件的选项才可用。选择"测试参数"、"操作参数"或"组件参数"作为输出类型时,通过"输出选项"对话框可以选择在其中存储运行会话持续时间的选定值的参数。

(3) 将值输出到数据表

选择"数据表"作为输出类型时,通过"输出选项"对话框可以指定在运行时数据表中存储选定值的位置,如图 6.44 所示。

在将值输出到数据表时,有以下选项可以修改。

- 名称。指定数据表中要存储值的列的名称。QTP 建议使用输出的默认名称。可以从列表中选择现有的输出名称,也可以通过使用默认输出名称或输入有效的描述性名称来新建输出名称。
- 数据表中的位置。输出测试的值时,指定将数据表列名称添加到数据表的全局工作表,还是当前操作工作表中。

(4) 将值输出到环境变量

如果选择"环境"作为输出类型时,通过"输出选项"对话框,可以指定要在其中存储运行会话持续时间选定值的环境参数,该参数由内部用户定义,如图 6.45 所示。

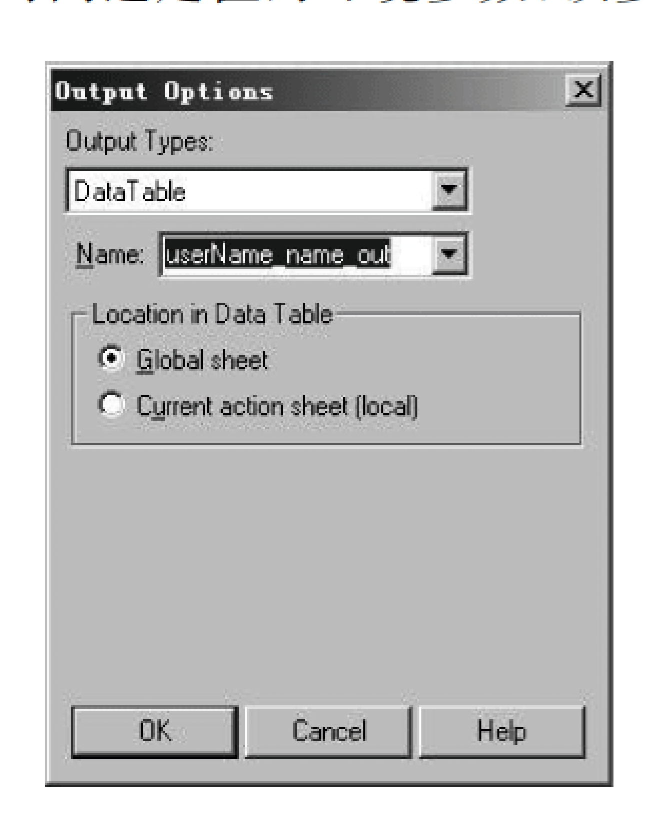


图 6.44　数据表输出类型

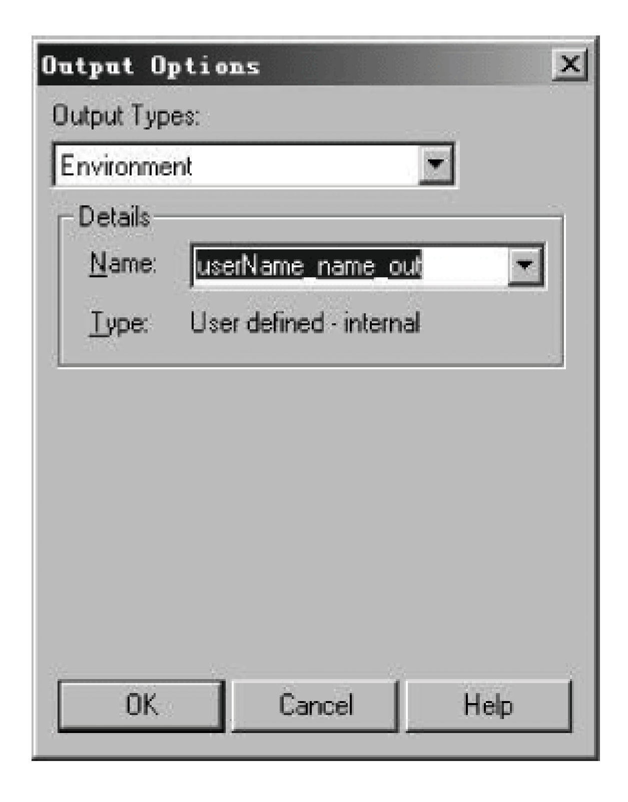


图 6.45　环境变量输出类型

4) 在脚本中建立输出值

在上例中,因为在表格检查点中机票价钱的预期结果并没有随着出发地点的改变而变动,导致第二、第三次的执行结果是失败的。

现在,我们从 Select a Flight: Mercury 网页上取得机票价钱,并且已取得的机票价钱更新表格检查点的预期结果,这样一来,测试脚本就可以利用在 Select a Flight: Mercury 网页上取得的机票价钱去验证 Book a Flight: Mercury 上显示的机票价钱。

首先,打开 Parameter 测试脚本,将脚本另存为 Output 测试脚本。

在树视图中,展开 Welcome: Mercury Tours 选项并且单击 Select a Flight: Mercury 网页,在 Active Screen 窗口会显示相应的页面。在 Active Screen 窗口中选取框住 270 后

右击,选择 Insert Text Output 命令,打开 Text Output Value Properties 对话框,如图 6.46 所示。

在 Text Output Value Properties 对话框中单击 Modify 按钮,打开 Output Options 对话框,如图 6.47 所示。

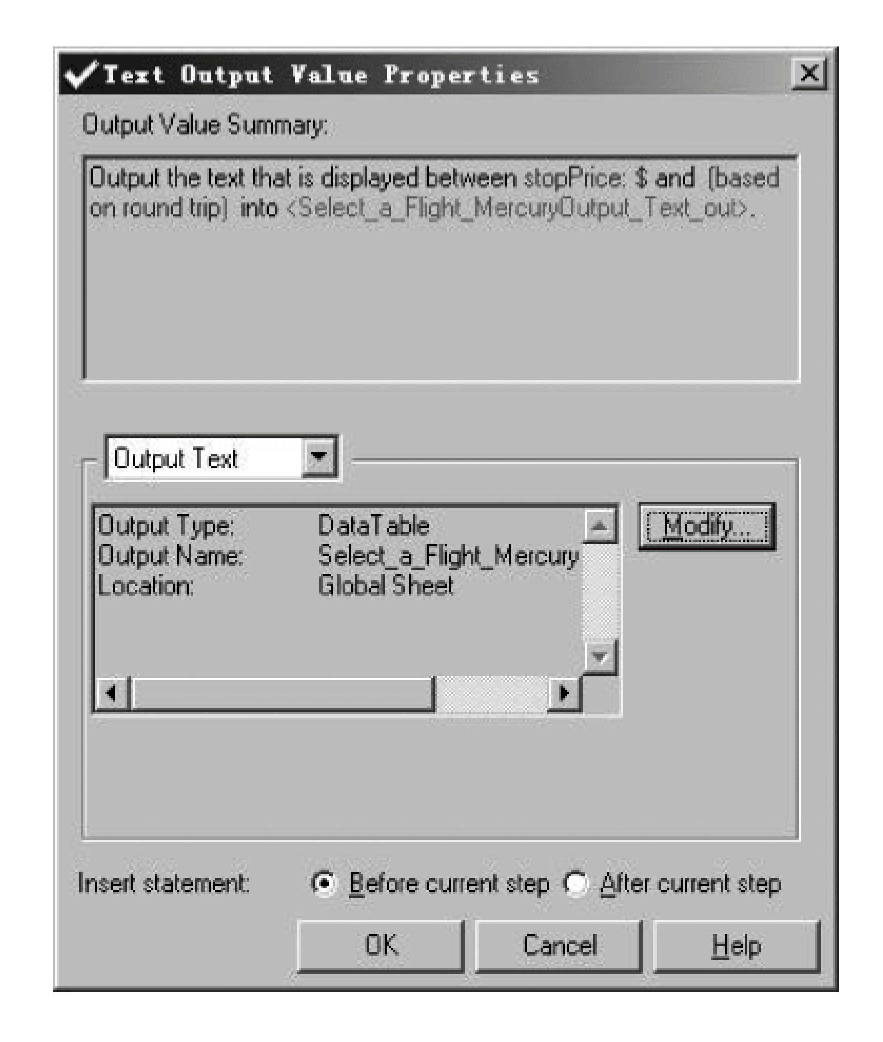


图 6.46　“文本输出值属性”对话框

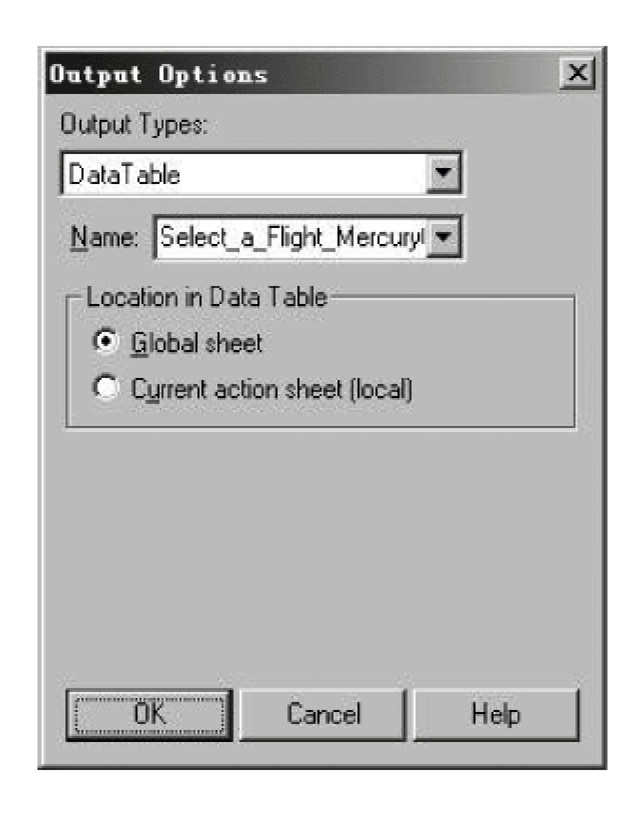


图 6.47　“输出选项”对话框

在“名称”下拉列表框中显示 Select_a_Flight_MercuryOutput_Text_out,将其更改为 depart_flight_price,接受其他默认值,单击 OK 按钮确认,QTP 会在 Data Table 中加入 depart_flight_price 字段。

在 Data Table 上的 depart_flight_price 字段的第一行会显示从应用程序上取得的输出值(270)。在执行时,第一次 QTP 会取得一样的值 270,接下来的第二、第三次会从应用程序上取得实际值,并存放在 Data Table 中。

修正表格检查点的预期值。在树视图中,展开 Welcome: Mercury Tours→Book a Flight: Mercury 目录,在 Checkpoint“New York to San Francisco”上右击,选择 Checkpoint Properties 命令,打开 Table Checkpoint Properties 对话框。

选中第三行、第三列(被勾选的字段),在 Configure value 选项组中选择 Parameter 选项,然后单击 Parameter Options 按钮,打开 Parameter Options 对话框,如图 6.48 所示。

在对话框的“名称”下拉列表中选择 depart_flight_price 选项。

单击 OK 按钮回到 Table Checkpoint Properties 对话框,可以看到这个检查点的预期结果已经被参数化了,如图 6.49 所示。

单击 OK 按钮关闭 Table Checkpoint Properties 对话框,保存测试脚本。

4. 执行并分析使用输出值的测试脚本

在上例中我们建立了输出值,并且将表格检查点参数化,现在,执行 Output 测试脚本。

单击工具栏上的 Run 按钮,开启 Run 对话框,选择 New run results folder 选项,其余为默认值,单击 OK 按钮开始执行脚本。当脚本运行结束后,会开启测试结果窗口。

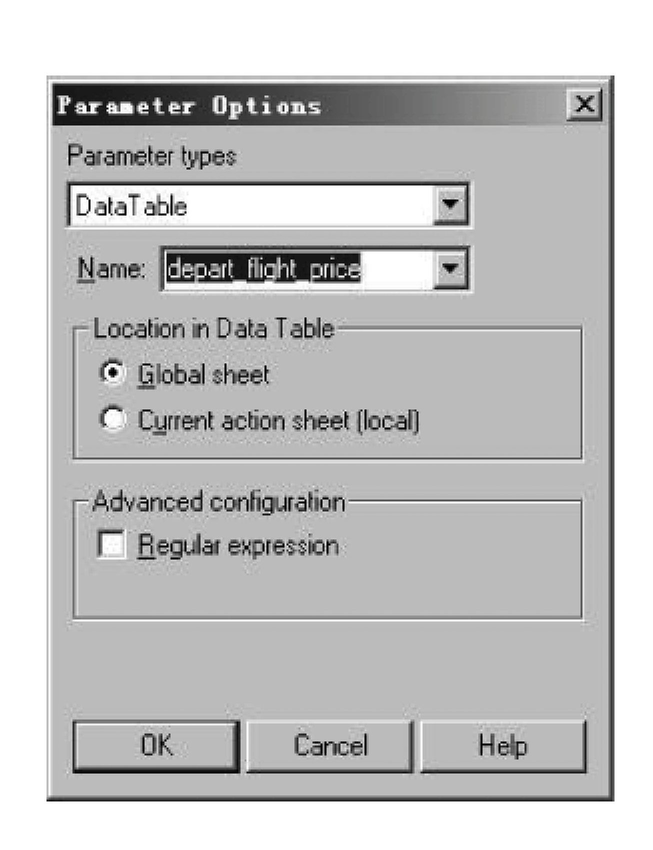


图 6.48 “参数选项”对话框

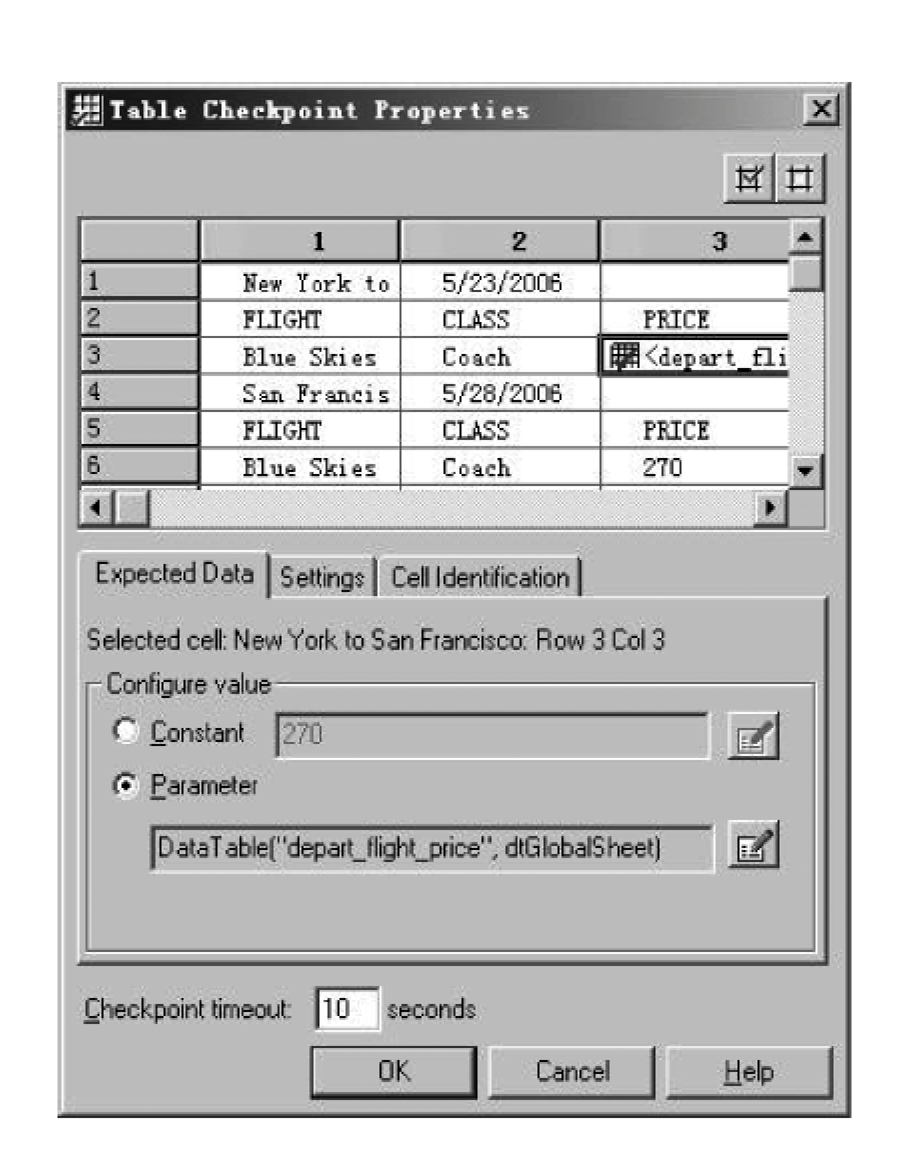


图 6.49 “表格检查点属性”对话框

在测试结果窗口中，单击树视图中的 Run-Time-Data 选项，可以在表格中看到执行测试时使用的输出值，在 depart_flight_price 字段中显示了不同的机票价钱，如图 6.50 所示。

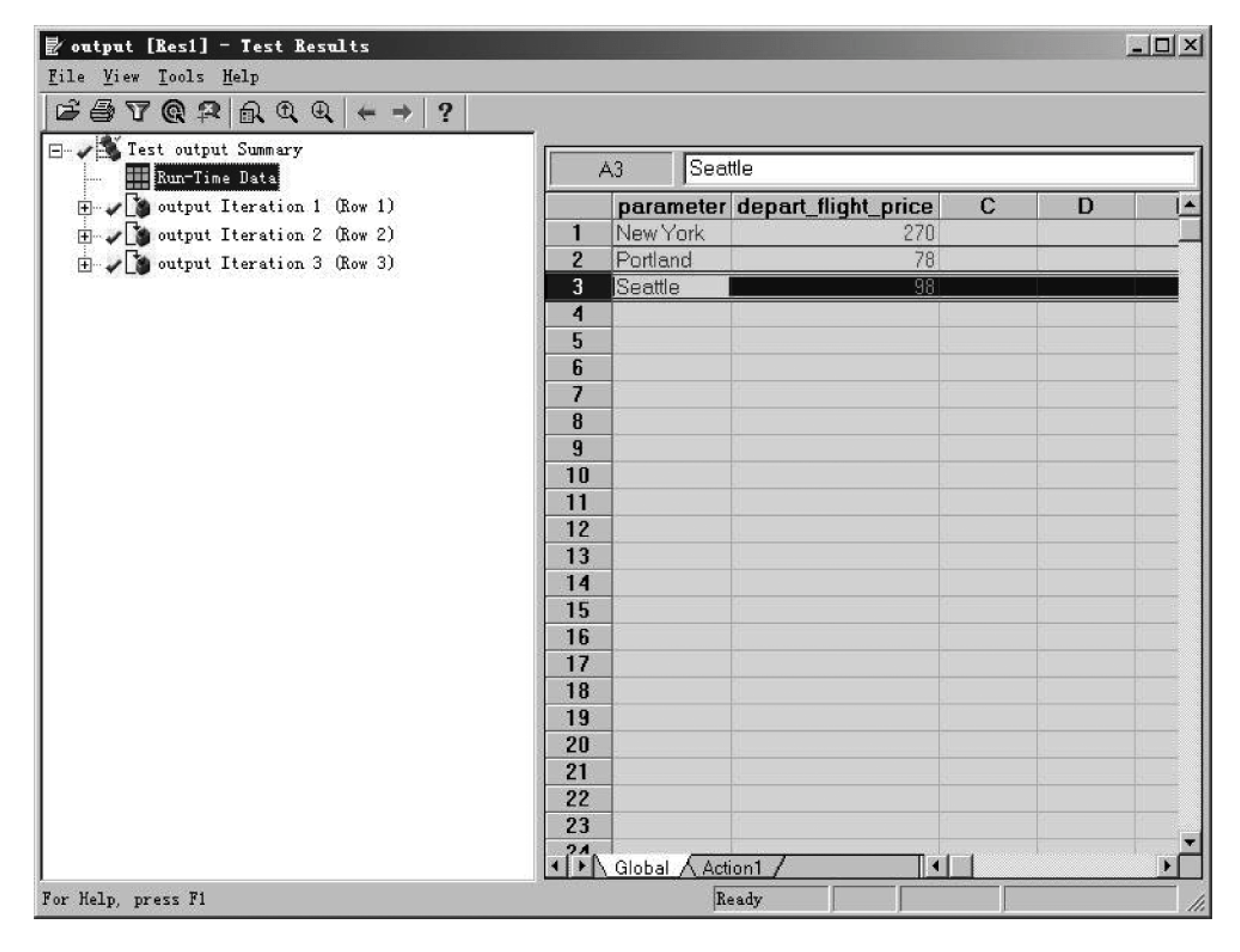


图 6.50 “测试结果”窗口

在结果窗口中单击 Test output Summary 选项可以看到，12 个检测点都通过了验证，运行结果均为 Passed。

6.2.2　能力目标

了解检查点的作用以及 QTP 中检查点的种类；掌握如何设置检查点以及验证检查点；掌握参数化设置步骤、参数种类、执行参数化测试脚本；掌握输出值类型、属性并在脚本中建立输出值。

6.2.3　任务驱动

1. 打开 QTP，并启动 Mercury Tours Web Site 系统。
2. 录制脚本 TestWeb。分别执行：使用用户名 Mercury、密码 Mercury 登录；输入订票日期(订票日期要大于系统当前时间)和出发城市、到达城市；查看航班信息并进行航班选择；输入订票用户姓名并预订航班；查看订单；注销等操作。
3. 进行运行时设置，并回放脚本。
4. 查看测试结果。

6.2.4　实践环节

1. 在 TestWeb 脚本中设置标准检查点、网页检查、文字/文字区域检查点和表格检查点。运行脚本，查看分析报告，验证检查点内容。
2. 对 TestWeb 脚本中的出发城市和到达城市进行参数化设置。

6.3　小　　结

- QTP 工作流程包括 5 个步骤，分别是：录制测试脚本；插入检查点；测试脚本参数化；运行测试脚本；测试结果分析。
- 检查点是将指定属性的当前值与该属性的期望值进行比较的验证点。这能够确定网站或应用程序是否正常运行。常用的检查点有：标准检查点、网页检查点、文字/文字区域检查点以及表格检查点。
- 参数化测试脚本包括数据输入的参数化和检测点的参数化。参数种类包括：测试，操作或组件参数，数据表参数，环境变量参数，随机数字参数。
- 通过 QTP 可以检索测试或组件中的值，并将这些值作为输出值存储。此后，就可以检索这些值，并在运行会话的不同阶段使用该值作为输入。输出值类型包括：标准输出值、文本/文本区输出值、数据库输出值、XML 输出值。

习　题　6

1. QTP 的工作流程包括哪 5 个步骤？
2. 检查点的类型有哪些？
3. 测试数据参数化的优势是什么？

4. 按如下要求录制脚本、执行脚本并分析测试报告。

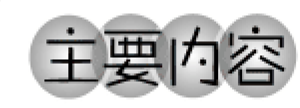
(1) 开启 Open Order 窗口，对"Order No."check box 建立标准检查点。

(2) 勾选"Order No."check box，输入订单编号。

(3) 对"Tickets："建立文本检查点。

(4) 单击 Exit 按钮退出系统。

第 7 章 LoadRunner 测试工具



- LoadRunner 的基本原理
- LoadRunner 的应用

本章将学习 LoadRunner 的基本工作原理，以及应用 LoadRunner 进行负载测试。LoadRunner 是一种预测系统行为和性能的工业标准级负载测试工具。通过模拟成千上万用户实施并发负载及实时性能监测的方式来确认和查找问题，LoadRunner 能够对整个企业架构进行负载测试。本章的重点包括如何使用 LoadRunner 进行用户脚本的录制，完善测试脚本，实施负载测试以及学会分析测试结果等。

7.1 LoadRunner 的基本原理

7.1.1 核心知识

目前企业的网络应用环境都必须支持大量用户同时在线访问，如购物网站和交友网站等。难以预知的用户并发访问量（即系统负载）和愈来愈复杂的应用环境使公司时时担心会发生用户响应速度过慢，系统崩溃等问题。这些问题都不可避免地导致公司收益的损失。Mercury Interactive 的 LoadRunner 是一种适用于各种体系架构的自动负载测试工具，它能预测系统行为并优化系统性能。LoadRunner 的测试对象是整个企业的系统，它通过模拟实际用户的操作行为和实时的性能监测，来帮助企业更快地查找系统的瓶颈并解决相关问题。

1. 核心组件

LoadRunner 包含 5 个核心组件，分别如下。

(1) Visual User Generator（虚拟用户生成器）。用于捕获最终用户业务流程和创建自动性能测试脚本（也称为虚拟用户脚本）。

(2) Controller（控制器）。用于组织、驱动、管理和监控负载测试。

(3) 负载生成器。用于通过运行虚拟用户生成负载。

(4) Analysis（分析器）。用来查看、分析和比较性能结果。

（5）Launcher（启动器）。为访问所有 LoadRunner 组件的统一界面。

2. 负载测试阶段

使用 LoadRunner 进行负载测试通常由 5 个阶段组成：制订测试计划，创建测试脚本，定义场景，执行场景和结果分析，如图 7.1 所示。



图 7.1　负载测试阶段

（1）制订测试计划。定义性能测试要求，例如并发用户的数量、典型业务流程和所需响应时间。

（2）创建测试脚本。将最终用户的活动捕获到脚本中。

（3）定义场景。使用 Controller 设置负载测试环境。

（4）执行场景。通过 Controller 驱动、管理和监控负载测试。

（5）结果分析。使用 Analysis 创建图和报告并评估性能。

3. 安装 LoadRunner

本教材中选用业界应用较为成熟的版本 LoadRunner 9.0。安装 LoadRunner 9.0 要求内存至少 512MB，硬盘空间 500MB。具体的安装步骤如下。

（1）进入欢迎页面，如图 7.2 所示。单击 Next 按钮。



图 7.2　LoadRunner 欢迎页面

（2）阅读相关软件许可协议，如图 7.3 所示。选择 I Agree 选项，并单击 Next 按钮。

（3）选择安装目录，如图 7.4 所示。单击 Browse 按钮可以改变安装目录，这里采用默认的安装目录，单击 Next 按钮。

（4）执行安装，如图 7.5 所示。单击 Next 按钮。

（5）成功安装页面，如图 7.6 所示。单击 Finish 按钮。



图 7.3　软件许可协议



图 7.4　选择安装目录

4. 负载测试运行环境

LoadRunner 支持 40 多种类型的应用程序的负载测试，本章只介绍基于 Web 的应用程序的负载测试，项目来自于 Mercury 企业解决方案中的一个基于 Web 的旅行代理系统 Mercury Tours。通过该系统用户可以连接到 Web 服务器、搜索航班、预订航班并查看航班路线等。

安装 LoadRunner 后，首先需要启动 Web 服务器，依次选择“开始”→“程序”→LoadRunner→Samples→Web→Start Web Server 命令。

接着，启动 Web 应用程序的首页面，依次选择“开始”→“程序”→LoadRunner→Samples→Web→HP Web Tours Application 命令。启动后的页面如图 7.7 所示。



图 7.5　执行安装



图 7.6　成功安装页面

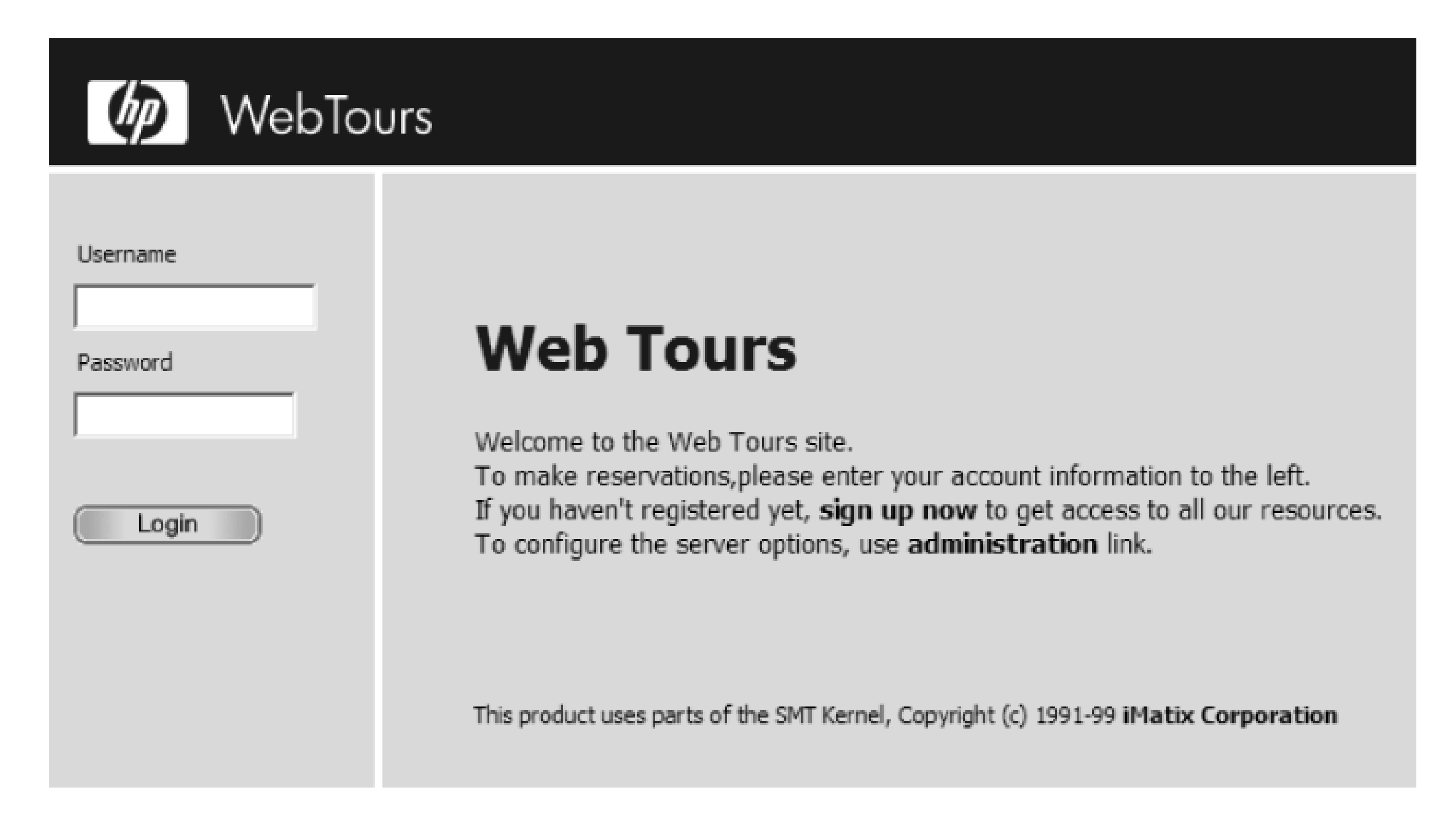


图 7.7　负载测试运行环境启动页面

该系统自带的用户名为 jojo，密码为 bean。在 Username 文本框中输入 jojo，在 Password 密码框中输入 bean，单击 Login 按钮，就可以进入到 Web Tours 系统的主页面，如图 7.8 所示。

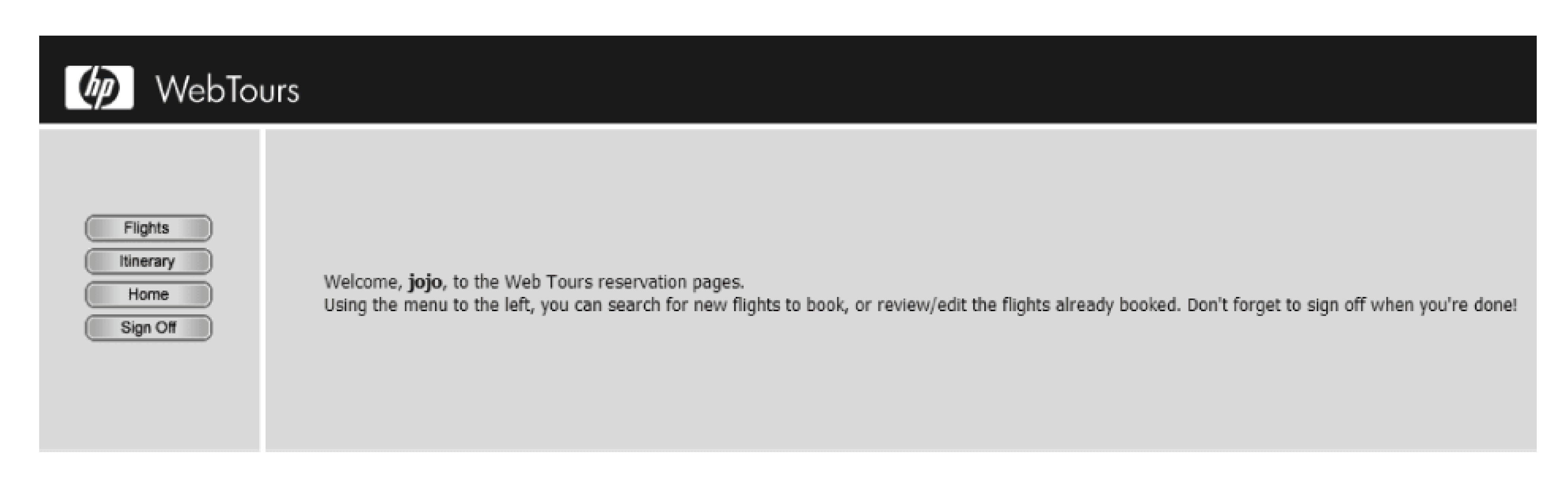


图 7.8　负载测试运行环境主页面

主页面的左侧为系统菜单，分别如下。

- 单击 Flights 按钮，进入航班管理页面。
- 单击 Itinerary 按钮，进入旅程管理页面。
- 单击 Home 按钮，重新回到主页面。
- 单击 Sign Off 按钮，用户退出该系统，进入登录页面。

主页面的右侧为显示区域，当单击左侧对应按钮时，右侧会显示具体相关的内容。

为了说明 LoadRunner 的功能，本章案例将针对 10 个并发用户的数据库应用程序运行和分析负载测试。该测试将模拟旅行代理同时使用航班预订系统进行如登录、搜索航班、购买机票、查看路线和注销等功能的负载测试。测试过程中，将使用 LoadRunner 的联机监控器观察 Web 服务器在该负载下的行为。尤其可以看到负载的增加将如何影响服务器对用户操作的响应时间（事务响应时间）以及如何引起错误。

5. 创建负载测试

使用 Controller（控制器）可以运行用来模拟实际用户执行操作的示例脚本，并可以通过让多个虚拟用户同时执行这些操作来在系统中创建负载。具体步骤如下。

（1）启动 LoadRunner 窗口

选择“开始”→“程序”→LoadRunner→LoadRunner 命令，启动如图 7.9 所示的 LoadRunner 窗口。

（2）录制脚本

要创建系统负载，需要首先生成模拟实际用户行为的自动脚本。在测试环境中，LoadRunner 会在物理计算机上用虚拟用户（即 Vuser）代替实际用户。Vuser 通过以可重复、可预测的方式模拟典型用户的操作，在系统上创建负载。LoadRunner 虚拟用户生成器（Virtual User Generator，VuGen）采用录制并播放机制。当用户在应用程序中按照业务流程操作时，VuGen 将这些操作录制到自动脚本中，以便作为负载测试的基础。选择 Create/Edit Scripts（创建或编辑负载脚本）选项，打开虚拟用户生成器，单击 New Vuser Script...按钮创建一个新虚拟用户脚本，如图 7.10 所示。

选择“单脚本协议类型”。协议是客户端用来与系统后端进行通信的标准。这里是对 Web 应用程序进行负载测试，选择 Web[HTTP/HTML]选项，并单击 OK 按钮，如图 7.11 所示。

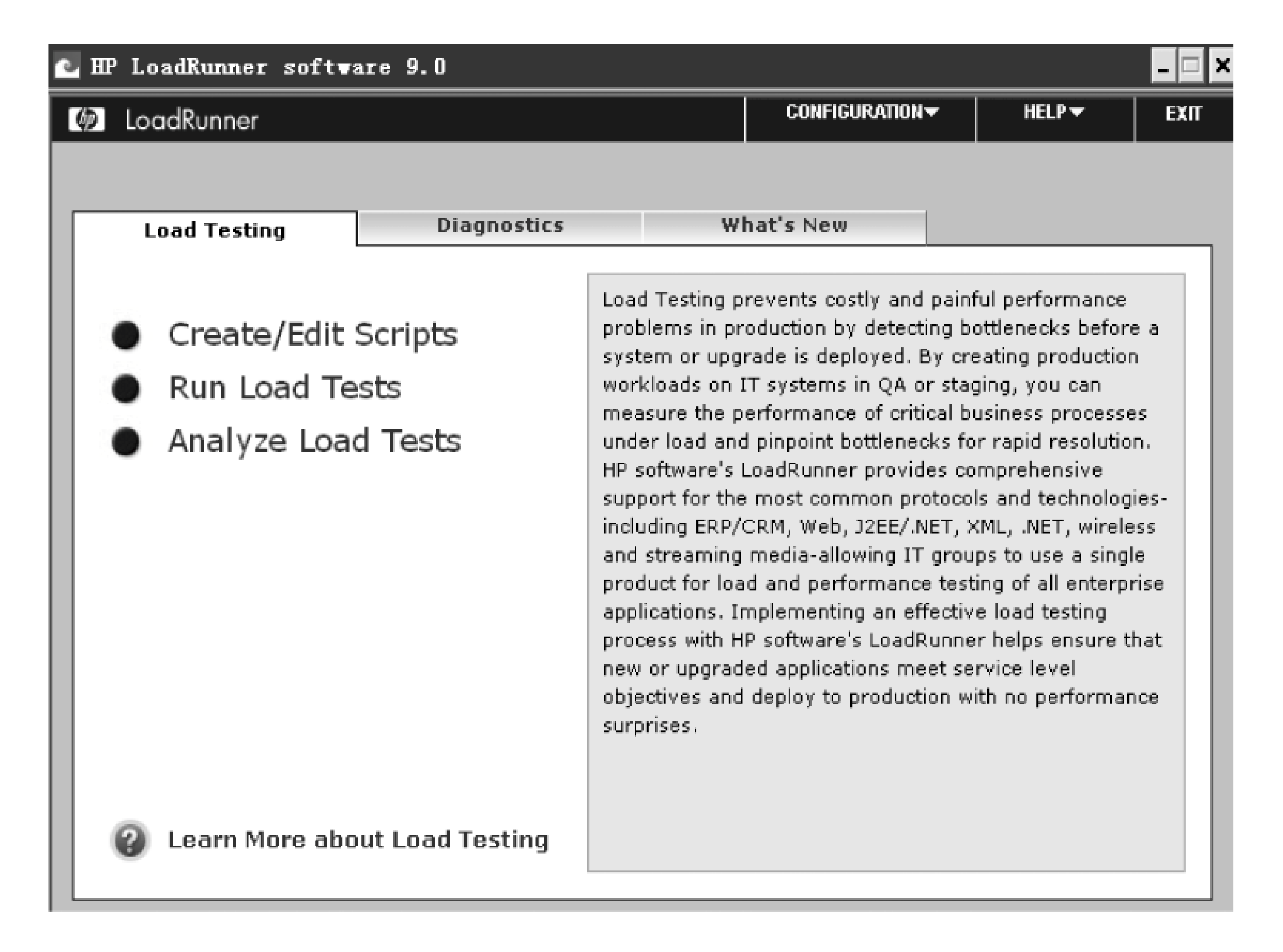


图 7.9　LoadRunner 启动窗口

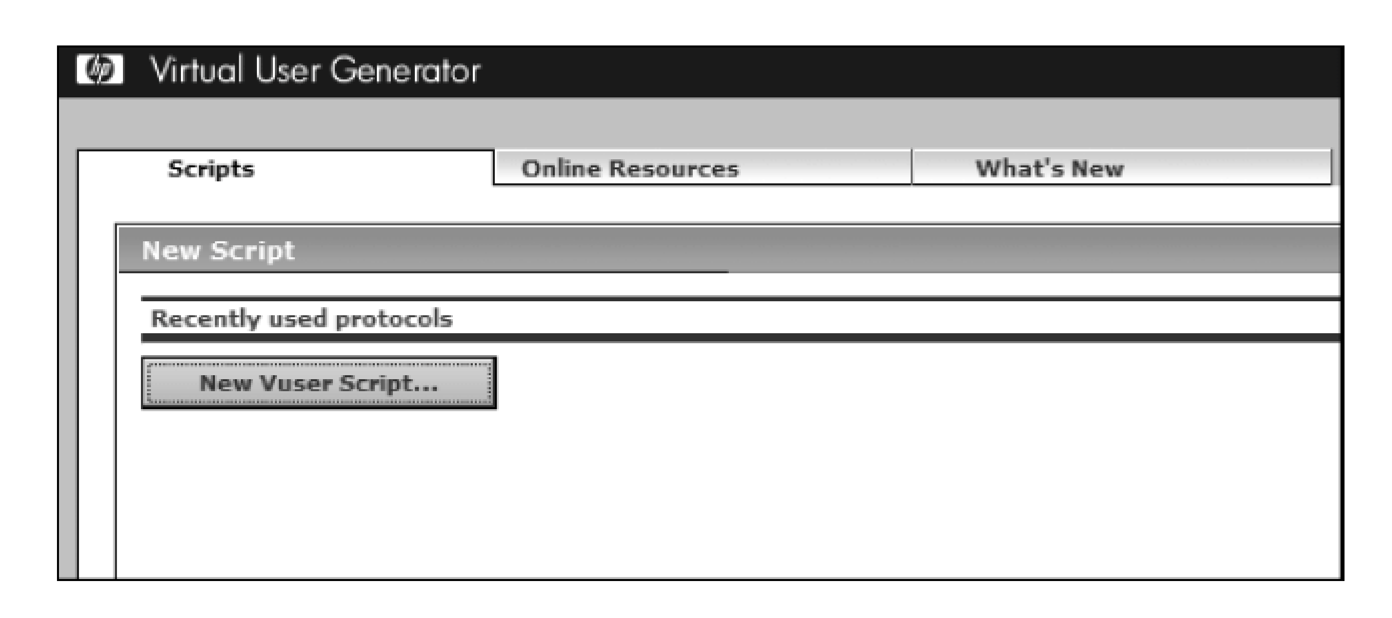


图 7.10　创建一个新虚拟用户脚本

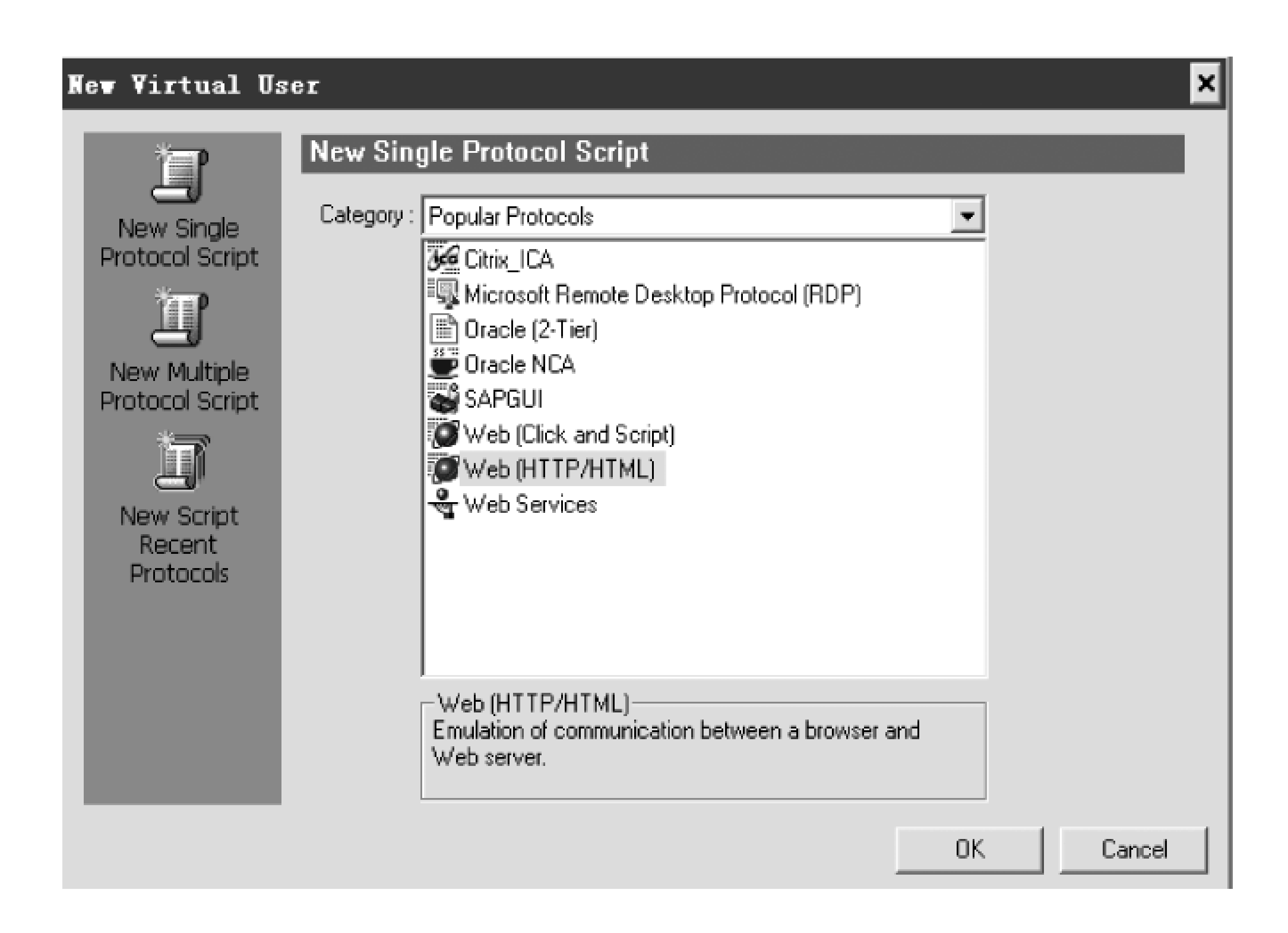


图 7.11　选择通信协议

开始录制脚本。单击 Start Record 按钮打开录制页面。在录制页面的 URL Address 栏中录入 Web 应用程序的网址 http://127.0.0.1:1080/WebTours/，单击 OK 按钮，这时，会自动打开旅游代理系统的首页，并开始录制用户脚本。首先进入登录页面，如图 7.12 所示。

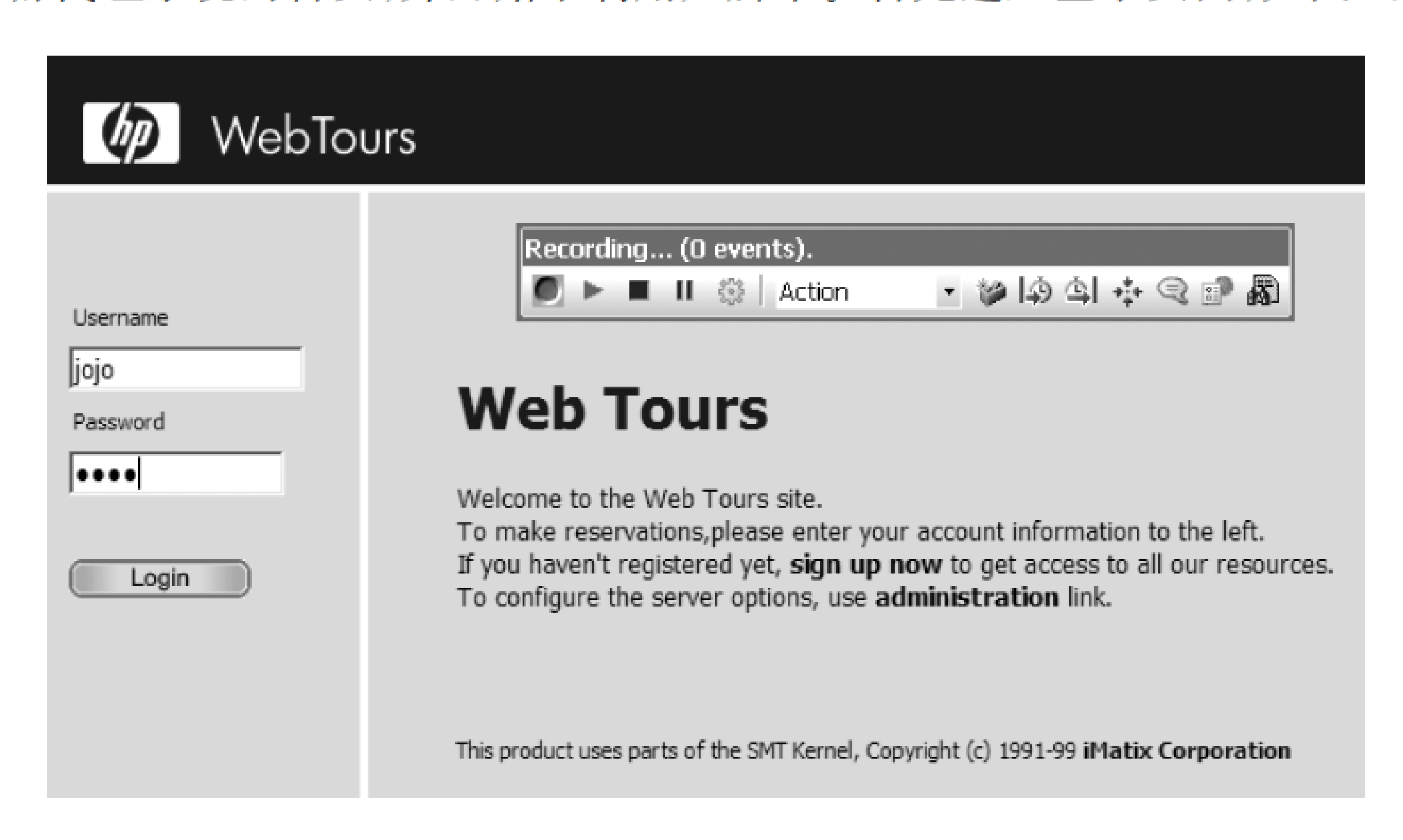


图 7.12　登录页面

对航线的基本要求进行选择，如图 7.13 所示，然后单击 Continue 按钮。

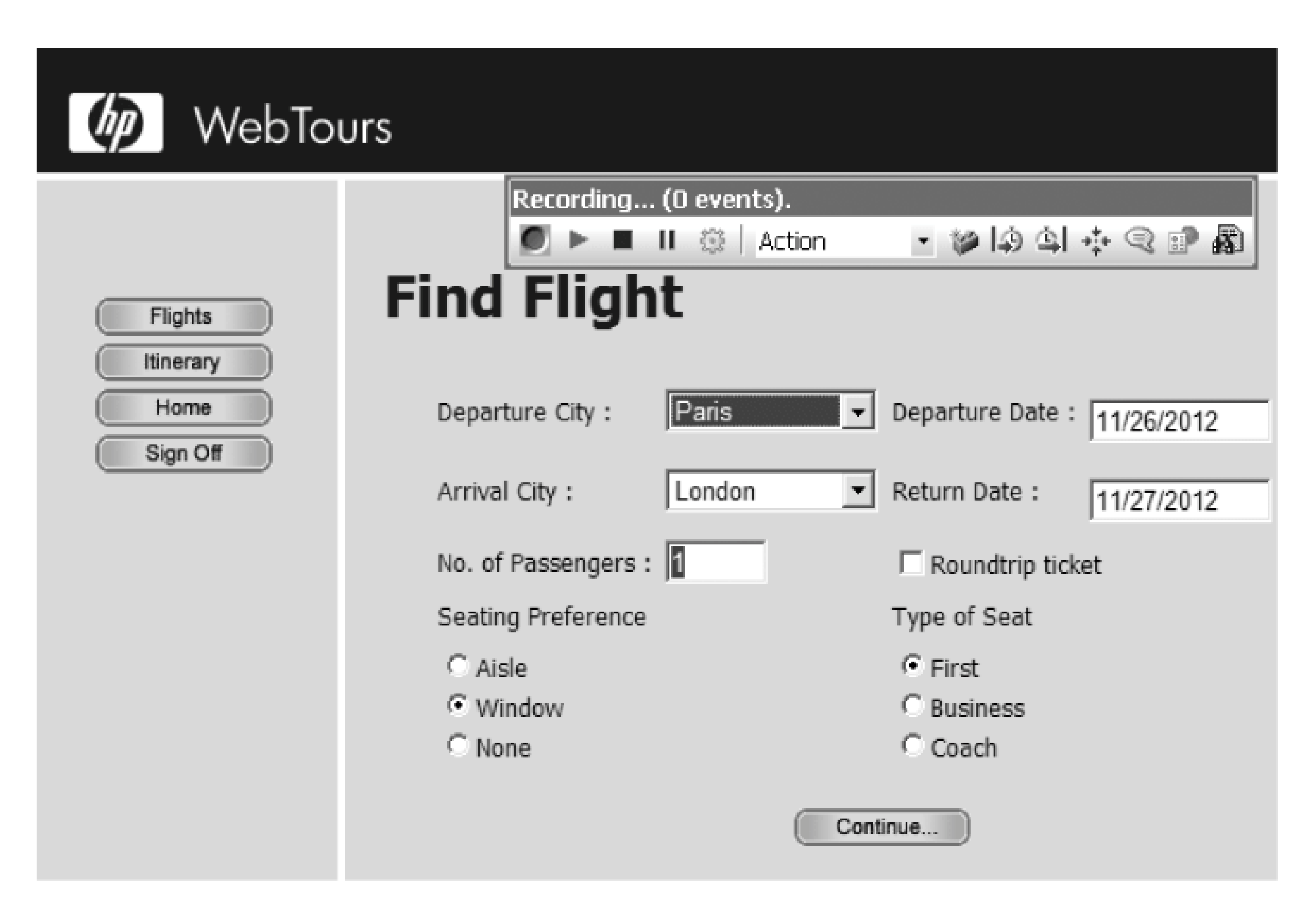


图 7.13　航线基本信息选择页面

显示满足条件的航线，选择后单击 Continue 按钮，如图 7.14 所示。



图 7.14　选择航线页面

选择付款信息，然后单击 Continue 按钮，如图 7.15 所示。

Payment Details

First Name : Joseph
Last Name : Marshall
Street Address : 234 Willow Drive
City/State/Zip : San Jose/CA/94085
Passenger Names : Joseph Marshall
Total for 1 ticket(s) is = $ 167
Credit Card : 12345678 Exp Date : 11/11
Save this Credit Card Information
Continue...

图 7.15　付款页面

显示清单页面，如图 7.16 所示。

Invoice

Thank you for booking through Web Tours.

JosephMarshall's Flight Invoice　Sun Nov 25 17:43:24 2012

Flights Reservation Requests:		Cost
1.	A First Class ticket from Paris to London. 11/26/2012 : 11pm : Flight 423 leaves Paris for London.	$167
*	Total Cost	$167
-	Total Charged to Credit Card # 12345678	$167
*	Credit Account Balance	$0.00

Book Another

图 7.16　清单页面

单击 Itinerary 按钮，查看旅程信息，已经包含了刚才预定的机票，如图 7.17 所示。



图 7.17　查看旅程信息页面

单击 Sign Off 按钮，退出该系统。选择菜单栏中的 Vuser→Stop 命令停止脚本的录制。

(3) 查看脚本

现在已经录制了旅行代理(包括登录、预订航班和注销等操作)。VuGen 录制了从单击 Start Record 按钮到单击 Stop 按钮之间所执行的步骤。可以通过两种方式查看脚本，分别是树视图和脚本视图。

通过单击图标 Tree 打开树视图。对于录制期间所执行的每一步骤，VuGen 都在测试树中生成一个图标和一个标题。在树视图中，将看到作为脚本步骤的用户操作。大多数步骤都附带相应的录制快照。快照使脚本更易于理解，更易于在测试工程师之间共享，这是因为可以准确看到录制过程中录制了哪些屏幕，可以随后进行快照比较以验证脚本的准确性。VuGen 还在回放期间创建每一步骤的快照，如图 7.18 所示。

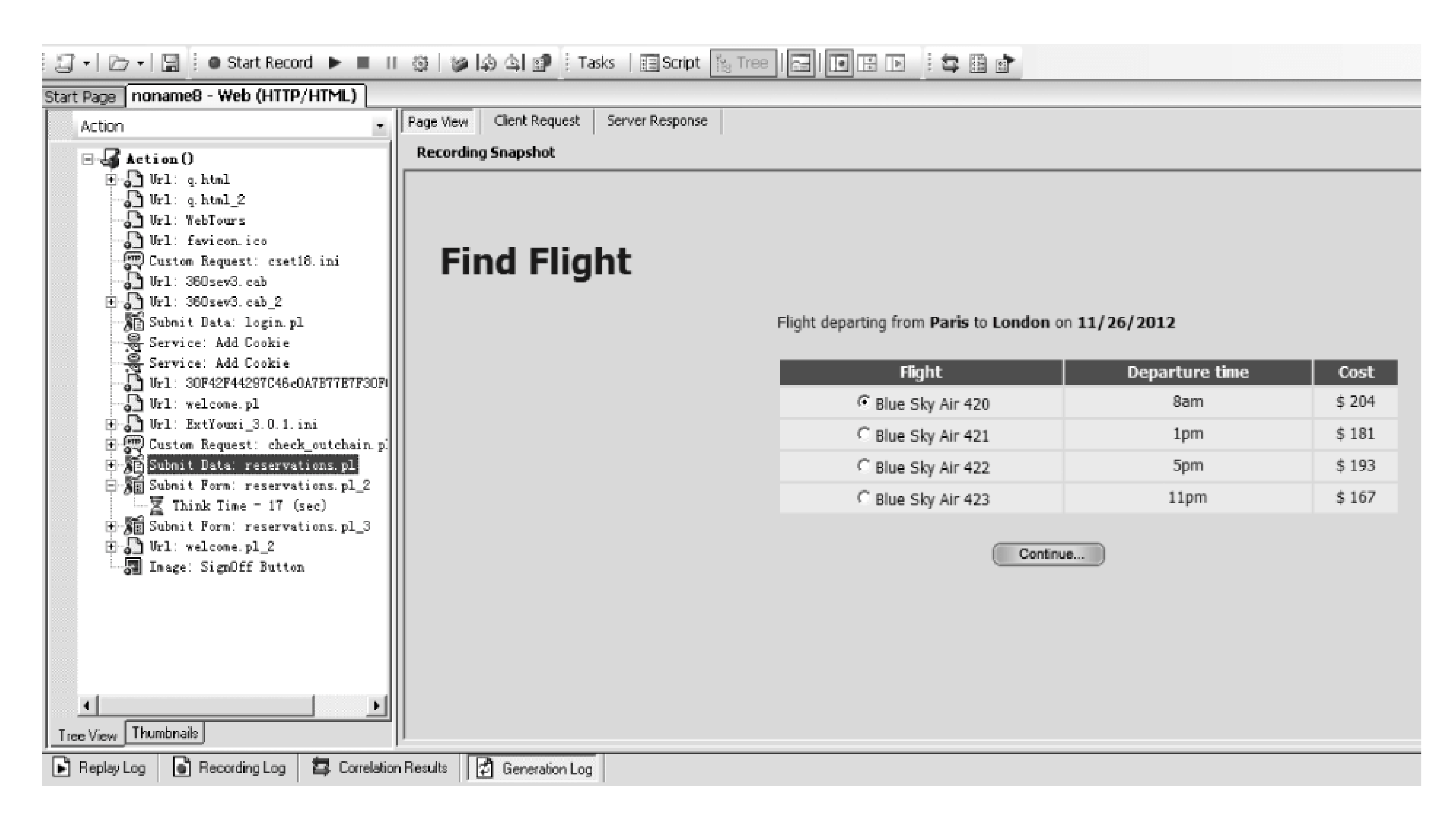


图 7.18　树视图

通过单击图标 Script 打开脚本视图。脚本视图是一种基于文本的视图，列出了作为 API 函数的 Vuser 操作。在脚本视图中，VuGen 将在编辑器中显示带有彩色编码的函数及其变量值的脚本。可以将 C 或 LoadRunner API 函数以及控制流语句直接键入此窗口中。脚本视图如图 7.19 所示。

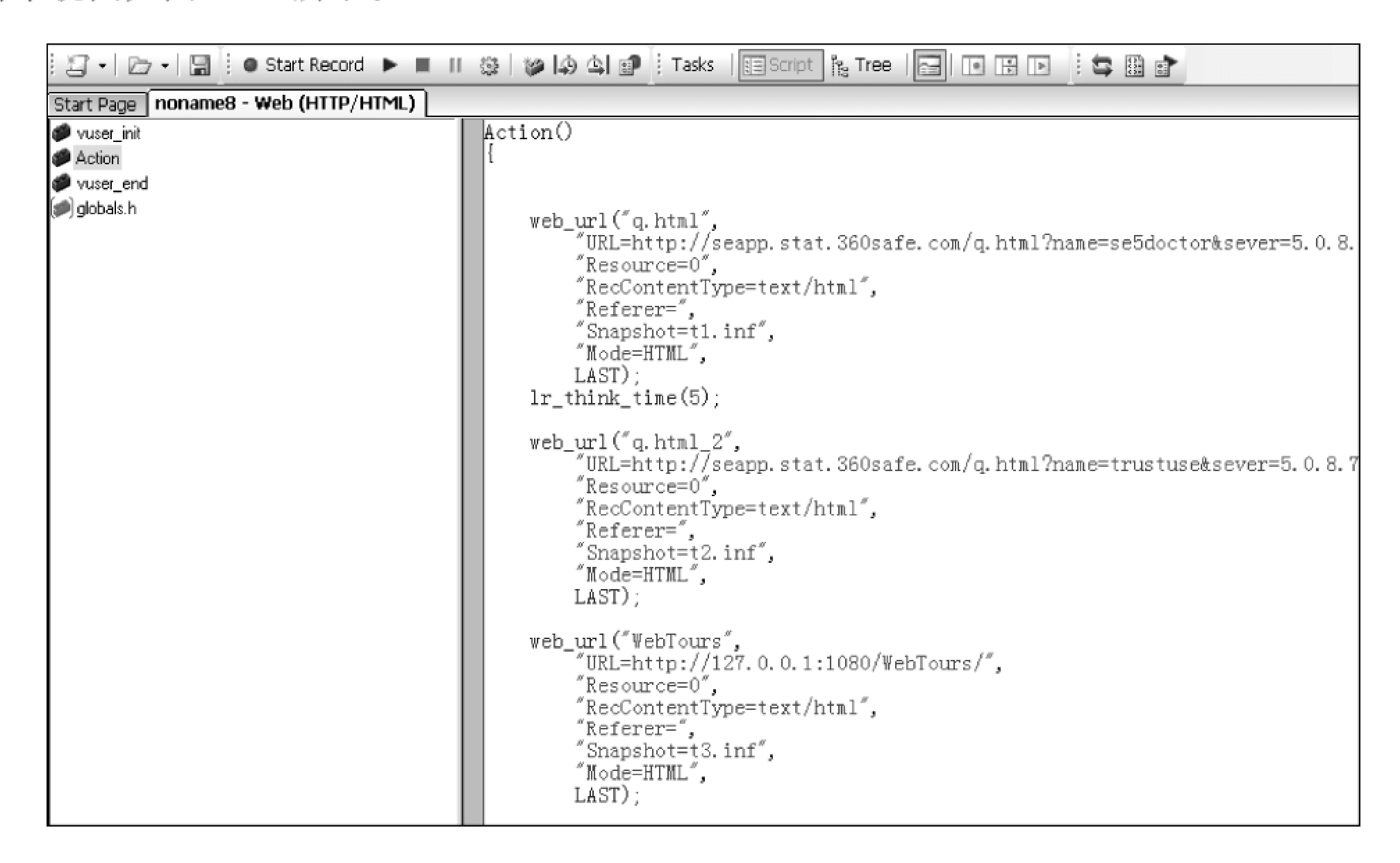


图 7.19　脚本视图

(4) 回放脚本

将脚本集成到负载测试场景中之前，回放脚本是必不可少的。回放已录制的脚本，可以用来验证该脚本是否能正常运行。回放期间，测试员可以在浏览器中查看已录制的操作并检查这些操作是否按照预期进行。

在回放脚本之前，可以配置运行时设置，这有助于设置 Vuser 的行为。通过 LoadRunner 运行时设置可以模拟各种实际用户的活动和行为。

例如，可以模拟对服务器的输出立即做出响应的用户，也可以模拟在每次做出响应之前先停下来思考的用户等。通过按 F4 键或单击工具栏中的“运行时设置”按钮，将打开“运行时设置”对话框，如图 7.20 所示。

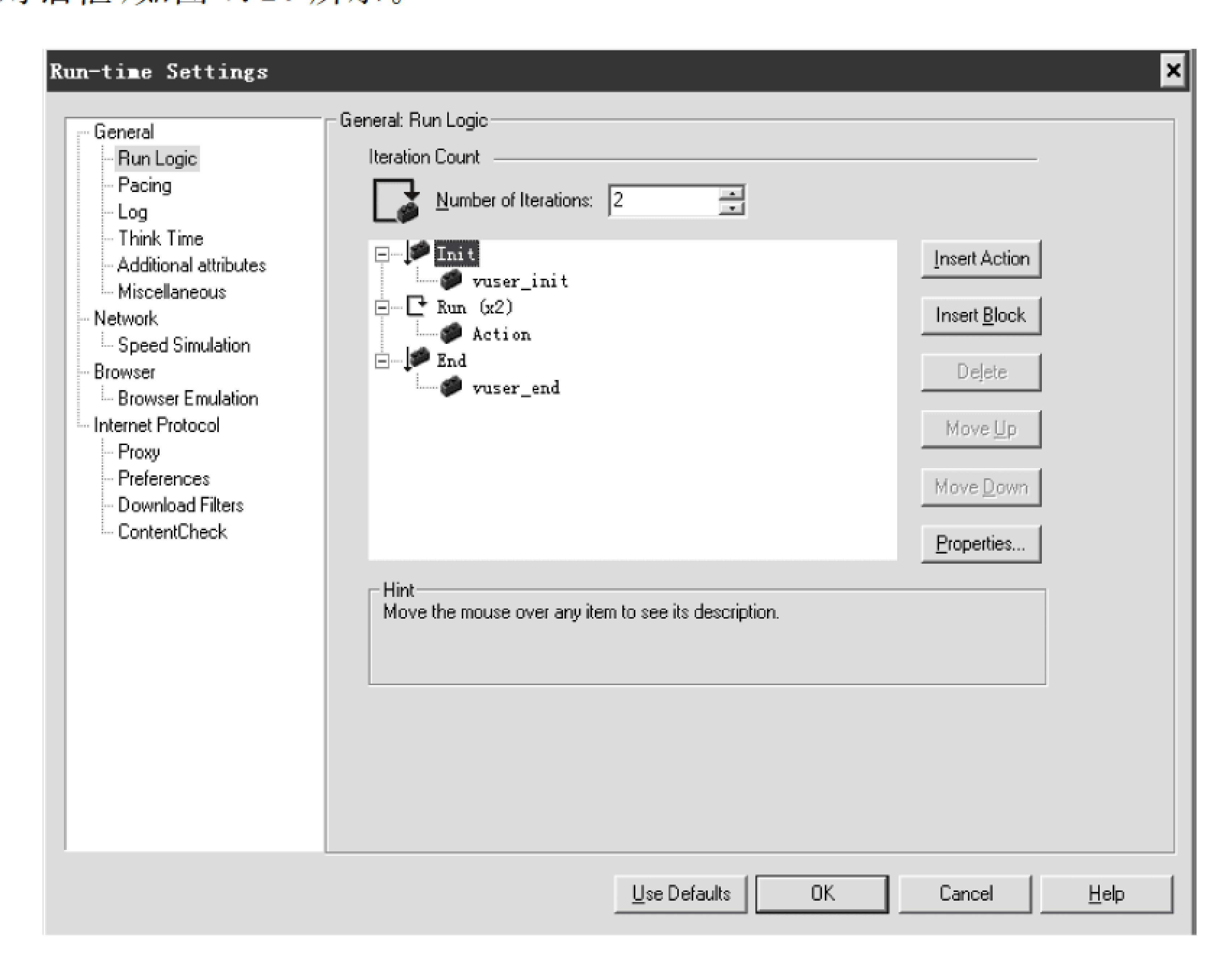


图 7.20 “运行时设置”对话框

运行时设置包括如下 4 部分内容。

- 运行逻辑（Run Logic）。主要用来设置脚本迭代的次数。
- 步（Pacing）。主要用来控制迭代之间的时间。可以将此时间指定为随机时间。
- 日志（Log）。主要设置测试记录的信息详细级别。开发期间，出于调试目的，可以选择启用某级别的日志记录，等验证脚本可以正常工作后，可以启用或禁用错误日志记录。
- 思考时间（Think Time）。步骤之间用户停止用以思考的时间。默认情况下脚本不包括思考时间，因此脚本将快速运行。

运行时设置完毕后就可以回放脚本了。VuGen 的运行时查看器功能将实时显示 Vuser 活动。默认情况下，VuGen 将在后台运行测试，而不显示脚本中操作的动画。可以选择 Tools→General Options→Display 菜单命令，勾选 Show browser during replay 选项，这样就可以在脚本回放时，浏览器中一并显示脚本录制的过程。按 F5 键或者 Run 按钮，进行脚本回放。

(5) 查看测试结果

回放录制的脚本后，需要查看脚本是否全部成功。如果某部分失败，则需要知道失败的原因和时间。可以通过选择 View→Test Results 命令查看测试结果，如图 7.21 所示。

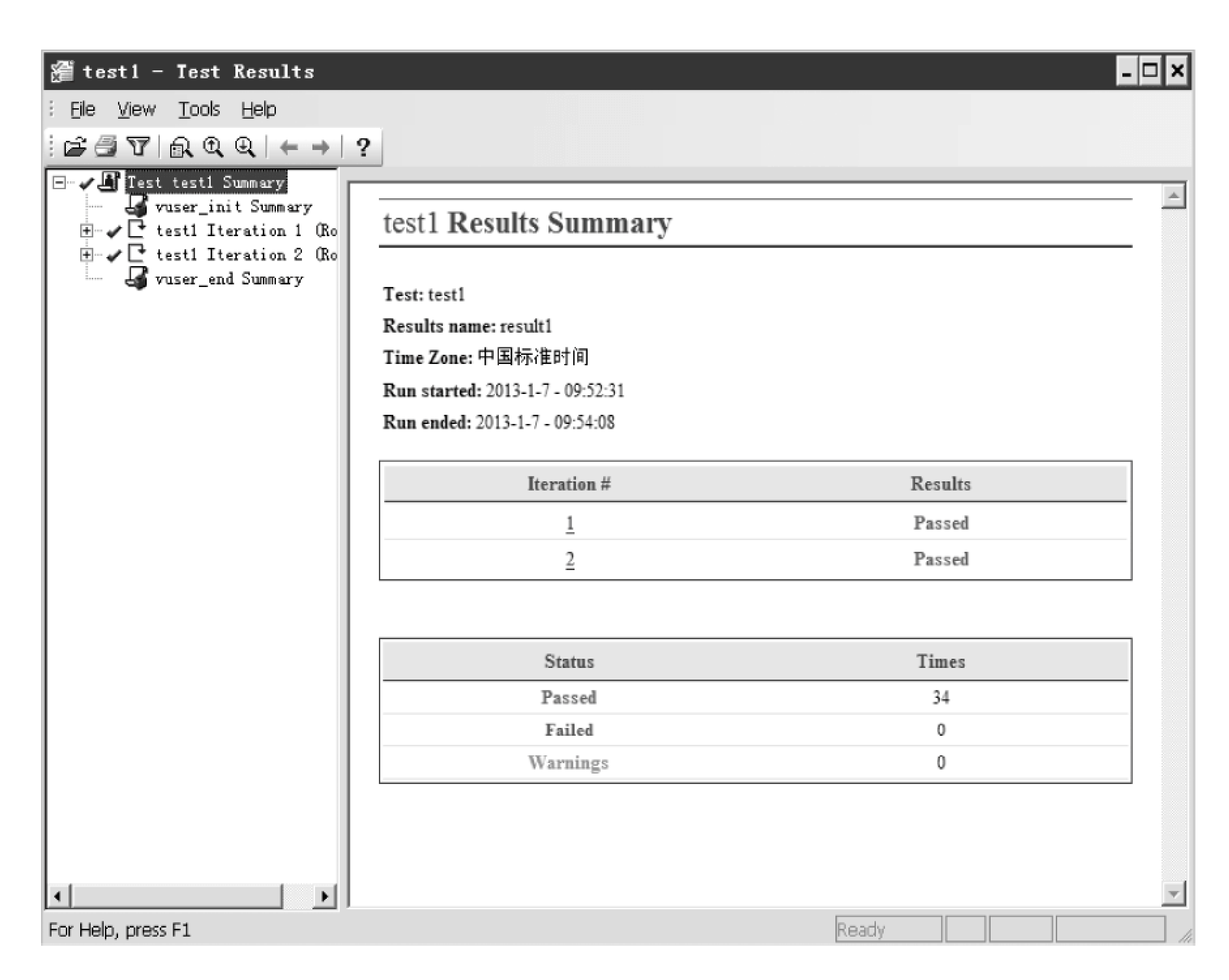


图 7.21　测试结果页面

7.1.2　能力目标

掌握 Load Runner 负载测试工具中的核心组件；掌握负载测试的主要流程；使用工具完成测试脚本的录制和回放；掌握运行时环境的设定以及对结果的分析。

7.1.3　任务驱动

1. 任务的主要内容

(1) 使用会员名 jojo、密码 bean 登录旅行代理系统 Mercury Tours。

(2) 单击“航班”选项，将打开“查找航班”页面，选择如下。

出发城市：丹佛(默认设置)

出发日期：保持默认设置不变(当前日期)

到达城市：洛杉矶

返回日期：保持默认设置不变(第二天的日期)

座位首选项：过道

保持其余的默认设置不变，然后单击“继续”按钮，将打开“搜索结果”页面。

(3) 单击“继续”按钮接受默认航班选择，将打开“付费详细信息”页面。

(4) 在“信用卡”文本框中输入 12345678，在“输出日期”文本框中输入 1212，单击“继续”按钮将打开“发票”页面，并显示您的发票。

(5) 单击左窗格中的“路线”选项，将打开“路线”页面。

(6) 单击左窗格中的“注销”选项。

2. 任务小结或知识扩展

这里可以进行机票的多次预订,按照任务驱动的方式进行修改就可以。也可以在后来录制用户脚本时,通过调整运行时的设置来改变脚本迭代的次数。

7.1.4 实践环节

1. 打开 LoadRunner,并启动网站服务器。

2. 录制脚本。分别执行:登录到 Mercury Tours 网站;输入航班详细信息;选择航班;输入付费信息并预订航班;查看路线;注销等操作。

3. 进行运行时设置,并回放脚本。

4. 查看测试结果。

7.2 LoadRunner 的应用

7.2.1 核心知识

在上一节中,已经在虚拟用户生成器(VuGen)中成功地录制了用户脚本,并通过回放脚本验证了测试脚本的正确性。在本节中,将重点讲解负载测试目标的确定,负载测试场景的建立,运行负载测试场景并对测试结果进行初步分析。

1. 负载测试目标

负载测试目标即负载测试是否有效的一个衡量标准。通常情况下需要确立五种不同类型的目标:并发 Vuser 数、每秒点击次数、每秒事务数、每分钟页面数和场景的事务响应时间。

当想了解可运行各种业务流程的 Vuser 总数时,可以使用虚拟用户目标类型。

当想知道服务器的稳定性,则可以使用每秒点击次数、每分钟页面数或每秒事务数目标类型。

当想知道所需的完成事务的响应时间,则可以使用事务响应时间目标类型。如,希望在 5s 内得到服务器的反馈信息。

这五个目标也有一定的关联和制约关系。如,当 Vuser 数不断增加时,必然会引起事务响应时间的增加。在目标的确定过程中,需要测试人员的经验。对待测服务器的处理能力以及性能有一定的初步判断后,确定负载测试目标。

2. 负载测试场景

负载测试场景即模拟负载的实际产生的情况。模拟的内容如下。

(1) 有多少用户同时在线访问待测系统,产生多重的负载?

(2) 这些负载是如何增加上去的?

(3) 加压完成时,能持续多长时间?

(4) 负载是如何消退的?

(5) 如何模拟不同用户的加压行为?

(6) 如何监控负载下的系统?

LoadRunner Controller 可以提供所有创建并运行负载测试场景的工具,以准确地模拟

系统实际的工作环境。

3. 测试结果分析

负载测试执行完成后，需要进行测试结果分析。结果分析的主要目的是查找出系统的性能故障，然后确定这些故障的根源。经常需要确认的问题有以下两种。

（1）结果是否满足了测试的预期目标？在负载下，用户终端的事务响应时间是多少？这些事务的平均事务响应时间是多少？

（2）系统的哪些部分导致性能下降？该网络和服务器的响应时间是多少？

LoadRunner Analysis 可以生成和查看图及报告，这将有助于测试人员找出性能问题并确定该问题的根源。

7.2.2　能力目标

掌握负载测试目标的确定；学会设计负载测试场景；进行测试结果的初步分析。

7.2.3　任务驱动

1. 任务的主要内容

（1）模拟 20 个旅行代理 Vuser 同时使用航班预订系统的操作。

（2）在负载测试之前对 20 个旅行代理 Vuser 进行初始化设置。加压前初始化 Vuser，可以减少 CPU 消耗并有助于提供更加真实的结果。

（3）设定每 15s 启动 2 个 Vuser 进行逐步加压过程。

（4）在 20 个 Vuser 都加压完成后，设定持续时间 3min。

（5）在减压过程中设定每 30s 停止 5 个 Vuser。

（6）观察负载测试过程中的性能数据。

（7）分析测试结果。

2. 任务小结或知识扩展

在负载测试时，进行性能数据的监测。如果各项数据比较平稳的话，说明系统可以正常承受该负载。这个时候，测试人员可以逐步增大 Vuser 的数量，并继续观察性能数据。如此往复，直到性能数据不满足预期目标时停止，系统负载测试可以结束。

7.2.4　实践环节

设计负载测试场景。根据任务驱动的主要内容设置负载测试场景。

（1）打开负载测试场景工具 LoadRunner Controller。选择“开始”→“程序”→LoadRunner→Applications→Controller 命令，如图 7.22 所示。

（2）选择 Manual Scenario（手动场景）选项。通过手动场景，可以控制正在运行的 Vuser 数量及其运行的时间，还可以测试应用程序可以同时运行的 Vuser 数。这里可以使用上节中录制的测试脚本，也可以使用 LoadRunner 自带的测试脚本。系统自带的测试脚本是在＜LoadRunner 安装目录＞\Tutorial 目录中的 basic_script 脚本文件。单击 Browse 按钮，选中 basic_script 脚本文件，再单击 OK 按钮，将打开 LoadRunner Controller 负载测试场景的主页面。在主页面中包括两个部分：Design 视图和 Run 视图。

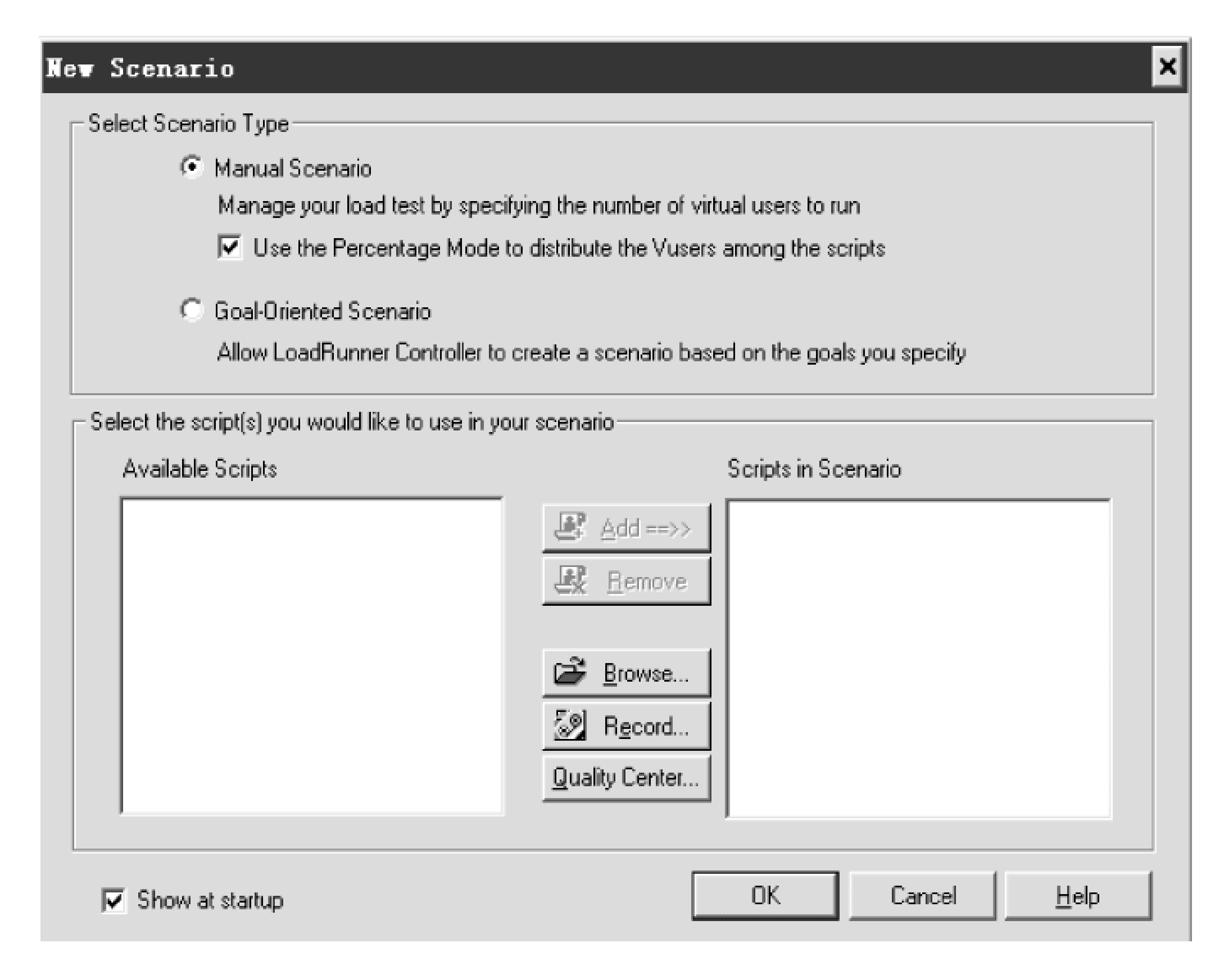


图 7.22　打开负载测试场景

Design 视图主要负责设计场景计划。包括：负载测试中 Vusers 的初始化；逐步加压的方式的设置；加压完成后，压力持续时间的设置；减压方式的设置。负载测试场景主页面如图 7.23 所示。

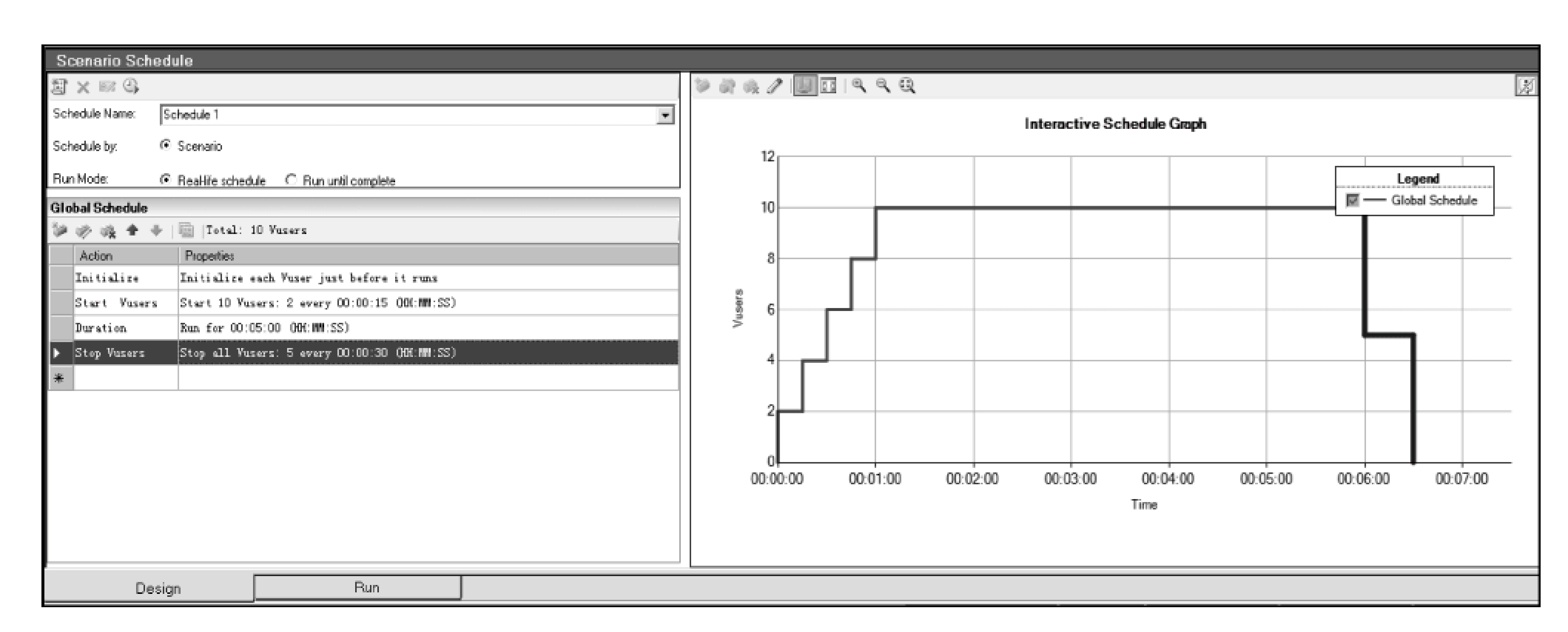


图 7.23　负载测试场景主页面

左侧为测试计划的设置，右侧为测试计划的图形化显示。这里只要在左侧对应的 Action 列中双击该选项，就可以对测试计划中的每一项进行设置。设置的内容参照任务驱动内容的要求，调整后的测试计划如图 7.24 所示。

Run 视图主要负责测试脚本的运行，以及系统性能的观测，如图 7.25 所示。

运行视图主要包括 5 个部分：Scenario Groups（场景组）、Scenario Status（场景状态）、Available Graphs（可用图树）、图查看区和图例。

- 场景组。位于左上窗格中，可以查看场景组中 Vuser 的状态。使用该窗格右侧的按钮可以启动、停止和重置场景，查看单个 Vuser 的状态，并且可以手动添加更多的 Vuser，从而增加场景运行期间应用程序上的负载。

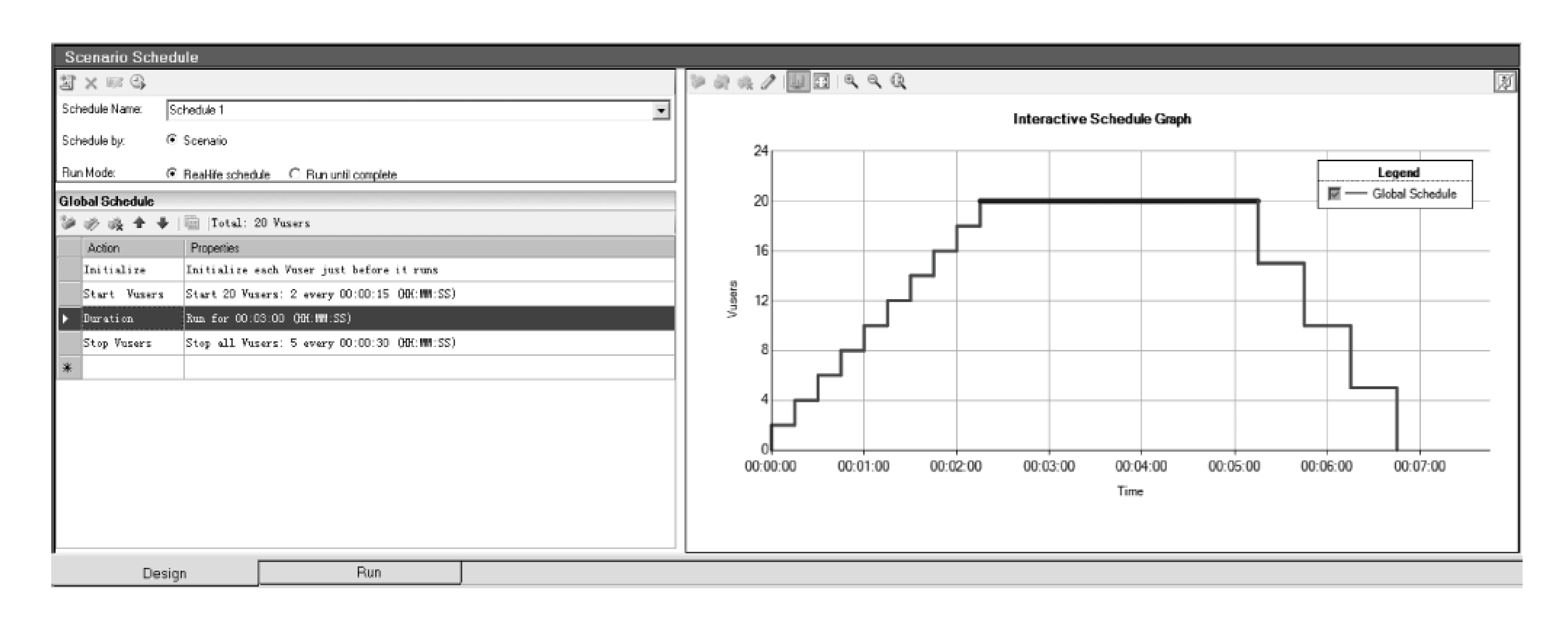


图 7.24　调整后的测试计划

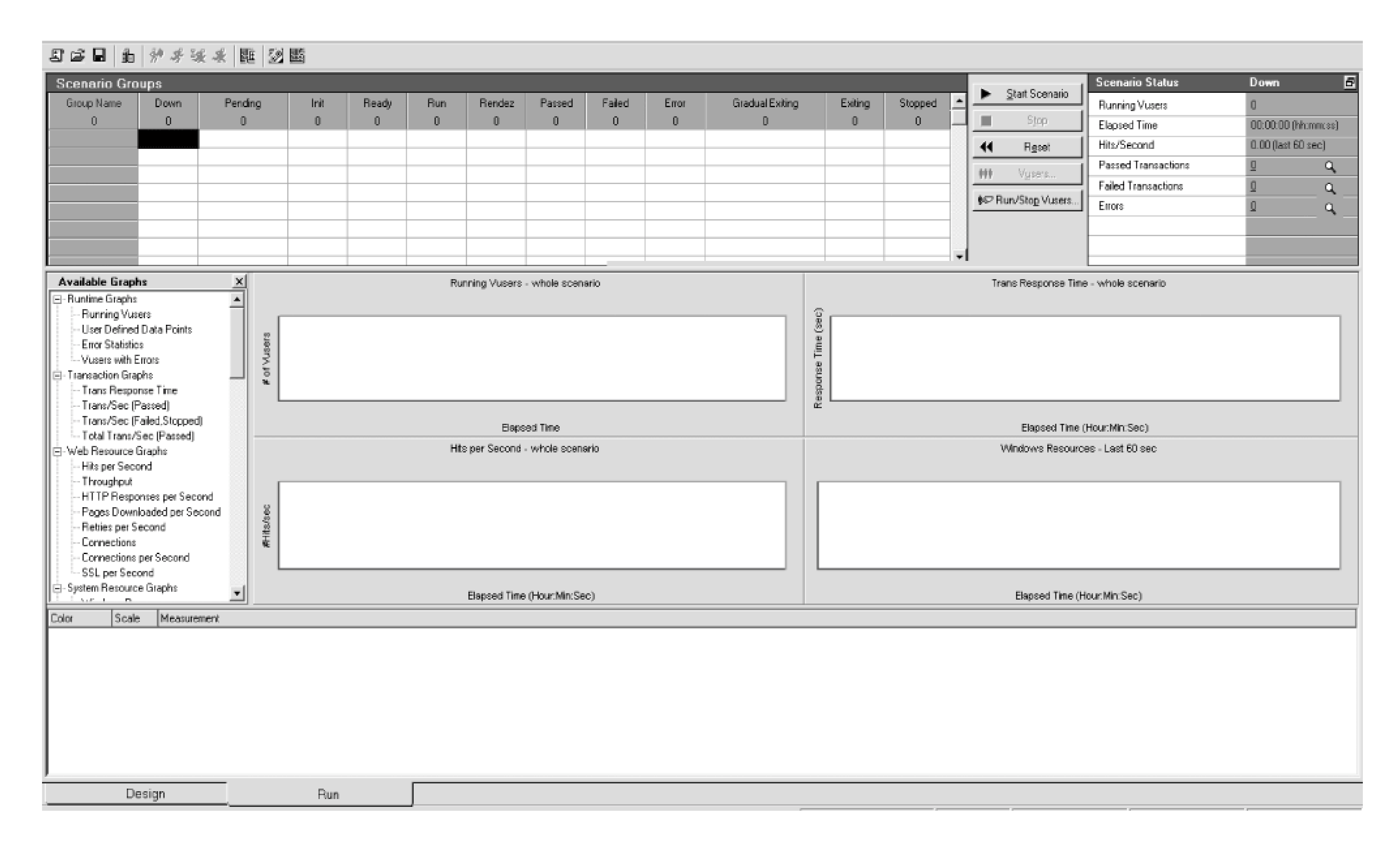


图 7.25　Run 视图

- 场景状态。位于右上窗格中，可以查看负载测试的概要，其中包括正在运行的 Vuser 数以及每个 Vuser 操作的状态。
- 可用图树。位于中部左侧窗格中，可以查看 LoadRunner 图列表。要打开图，则在该树中选择一个图，然后将其拖动到图查看区域中。
- 图查看区域。位于中部右侧窗格中，可以自定义显示，以查看 1～8 个图(选择 View→View Graphs 命令)。
- 图例。位于底部窗格中，可以查看选定图中的数据。

(3) 添加负载生成器。单击图 7.25 左上角第四个按钮 Load Generators，打开负载生成器页面，单击 Add 按钮，如图 7.26 所示。

在“添加负载生成器”页面中指定负载生成器的名字为 localhost，单击 OK 按钮，如图 7.27 所示。

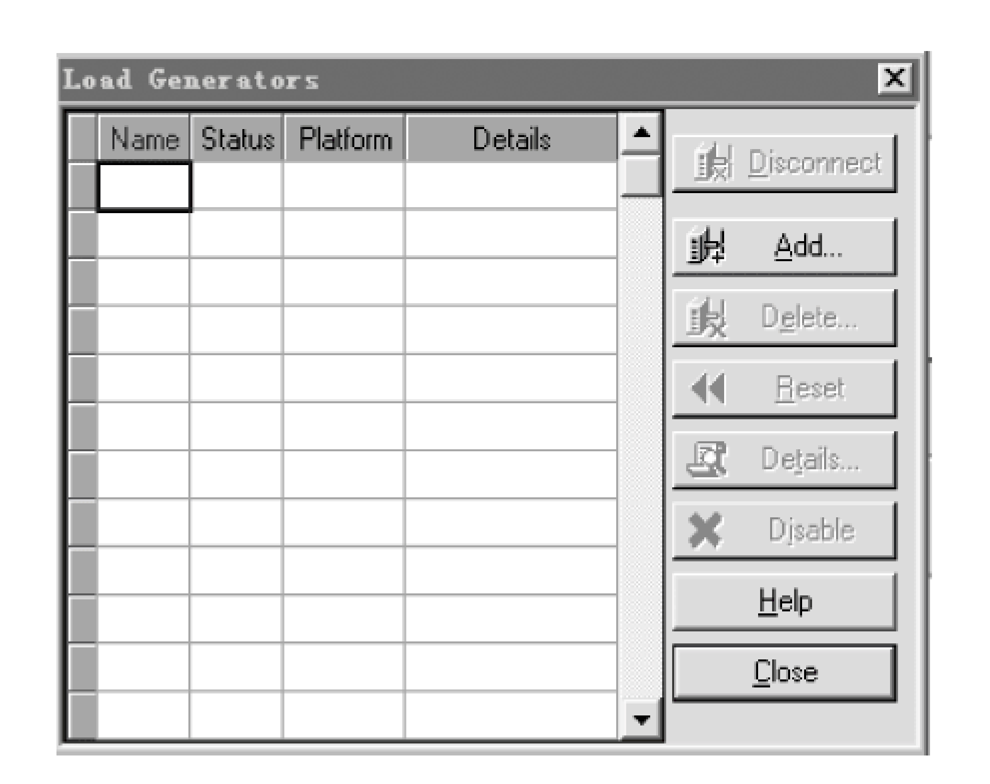


图 7.26 添加负载生成器

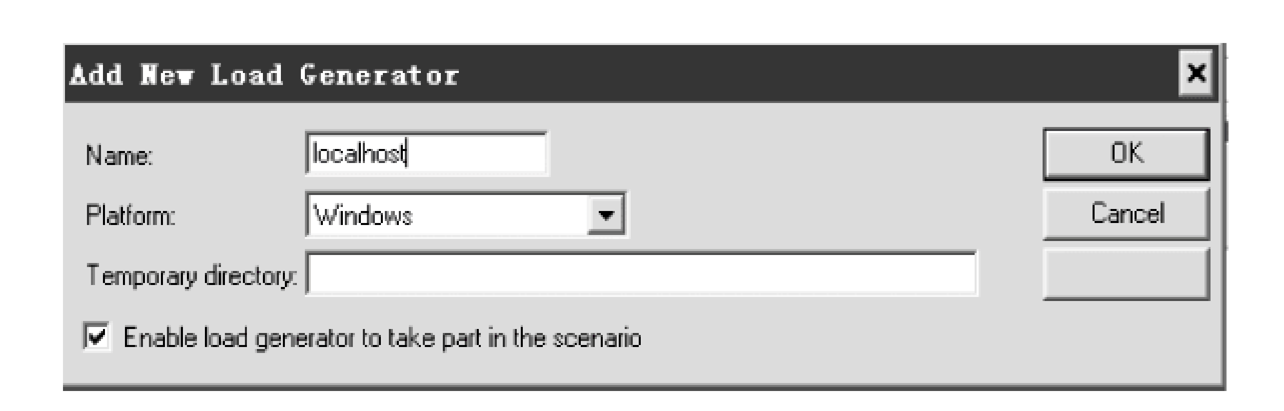


图 7.27 指定负载生成器

(4) 启动测试场景,进行负载测试,单击 Start Scenario 按钮。负载测试执行过程中,系统的相关参数指标也在发生改变,测试人员需要仔细观察和对比。默认情况下,主要观察中间四个视图的性能数据。四个视图分别如下。

- Running Vusers-whole scenario(正在运行的 Vuser—整个场景图)。该视图显示指定时间正在运行的 Vuser 数。
- Trans Response Time-whole scenario(事务响应时间—整个场景图)。该视图显示完成每个事务所需的时间。
- Hits per Second-whole scenario(每秒点击次数—整个场景图)。该视图显示场景运行的每一秒内 Vuser 在 Web 服务器上的点击次数(HTTP 请求数)。
- Windows Resources(Windows 资源图)。该视图显示场景运行期间度量的 Windows 资源。默认情况下,该视图不显示任何内容,需要在该视图上右击选择 Add Measurements 命令,在弹出的 Windows Resources 窗口中单击 Add 按钮,指定观测的服务器名称为负载生成器名称,即 localhost,再单击 OK 按钮,这个时候该视图以及图例区域就会显示相关的参数信息了,如图 7.28 所示。

(5) 负载测试完成以后,通过选择 Results→Analysis Results 菜单命令打开测试分析工具 LoadRunner Analysis。在 Analysis 会话过程中生成的图和报告提供了有关系统性能的重要信息。使用这些图和报告,可以轻松地标识和确定应用程序中的瓶颈以及提高系统性能所需的改进。负载测试的测试报告摘要如图 7.29 所示。

从报告摘要中可以知道该负载测试中的如下一些基本信息。

最大运行虚拟用户数:20

总吞吐量:162 792 369 字节

平均吞吐量(字节/秒):383 944

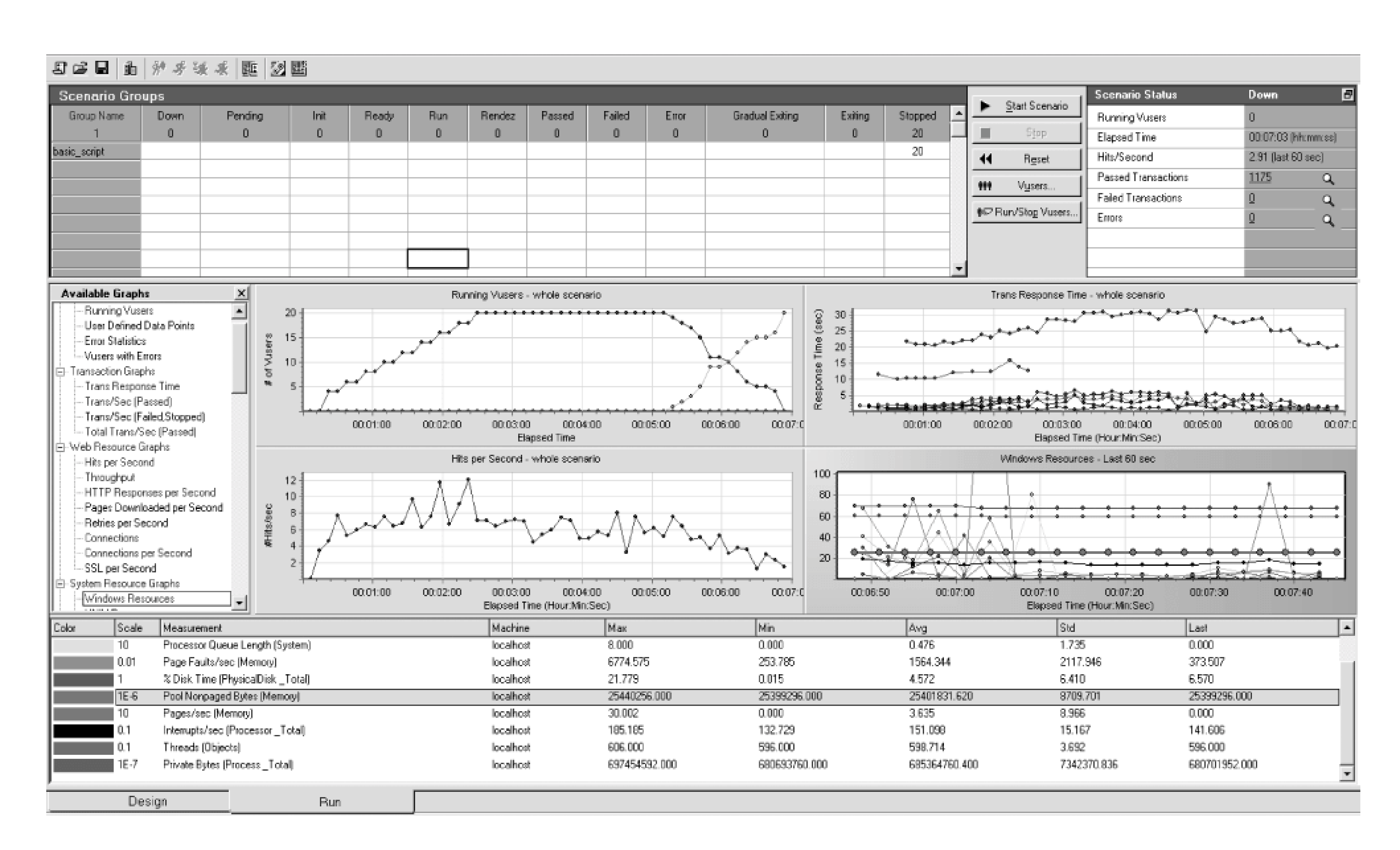


图 7.28　负载测试性能观测视图

Analysis Summary　　Period: 25/01/2013 15:21:25 - 25/01/2013 15:28:28

Scenario Name:　Scenario1
Results in Session: C:\Documents and Settings\Administrator\Local Settings\Temp\res\res.lrr
Duration:　7 minutes and 3 seconds.

Statistics Summary

Maximum Running Vusers:	20
Total Throughput (bytes):	162,792,369
Average Throughput (bytes/second):	383,944
Total Hits:	2,435
Average Hits per Second:	5.743　View HTTP Responses Summary

图 7.29　负载测试的测试报告

总的页面点击数：2435

平均每秒点击数：5.743

在测试报告中，最重要的是事务概要。在事务概要中列出了有关每个事务行为的基本情况，如图 7.30 所示。

在本次测试场景中，所有事务全部通过，说明服务器能够承受 20 个旅行代理 Vuser 同时使用该航班预订系统。如果某个事务失败的次数较多的话，可以重点分析该事务的基本情况。可以切换到 Average Transaction Response Time（平均事务响应时间）页面，找出事务的平均响应时间显著波动的点，把鼠标指针移到该点上，会提示在场景运行了多少时间时达到峰值。找到这个时间点后，切换到 Running Vusers 页面，得到该时间点上 Vusers 的个数，即 Vusers 的个数超过该值时，事务的响应时间会明显增长，从而得到服务器的 Vusers 的负载数量极值。

Transaction Summary

Transactions: Total Passed: 1,175 Total Failed: 0 Total Stopped: 0   Average Response Time

Transaction Name	SLA Status	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
Action_Transaction		19.589	26.972	33.305	3.922	31.094	215	0	0
S01_T01_HomePage		0.86	1.928	4.169	1.054	3.091	20	0	0
S01_T02_LogIn		0.701	1.874	3.831	1.029	3.733	20	0	0
S01_T03_SearchFlight		0.662	3.237	5.86	1.312	4.868	215	0	0
S01_T04_BookFlight		0.57	2.347	5.867	1.352	4.342	215	0	0
S01_T05_PaymentDetails		0.285	1.054	3.544	0.714	2.315	215	0	0
S01_T06_CheckItinerary		0.721	4.252	7.29	1.879	6.574	215	0	0
S01_T07_SignOff		0.568	1.916	3.783	1.218	3.623	20	0	0
vuser_end_Transaction		0.568	1.916	3.783	1.218	3.623	20	0	0
vuser_init_Transaction		9.958	11.84	15.851	1.678	13.413	20	0	0

Service Level Agreement Legend: Pass Fail No Data

图 7.30 事务概要

7.3 小 结

- LoadRunner 包含 5 个核心组件，分别是：Visual User Generator（虚拟用户生成器）、Controller（控制器）、负载生成器、Analysis（分析器）和 Launcher（启动器）。
- 使用 LoadRunner 进行负载测试通常由 5 个阶段组成：制订测试计划，创建测试脚本，定义场景，执行场景和结果分析。
- 负载测试目标即负载测试是否有效的一个衡量标准。通常情况下需要确立 5 种不同类型的目标：并发 Vuser 数、每秒点击次数、每秒事务数、每分钟页面数和场景的事务响应时间。

习 题 7

1. 简述负载场景如何设置，主要包括哪些核心步骤。

2. 举例说明负载测试目标包括哪几项。

3. 针对自己的 Web 项目，使用 LoadRunner 进行一个用户操作脚本的录制以及回放该脚本。

4. 针对自己的 Web 项目，指定负载测试目标，进行测试场景的设置，并执行负载测试，观察测试结果，找出项目中的瓶颈，并确定待测系统服务区的最大 Vusers 数。

第 8 章

JUnit 测试工具

主要内容

- JUnit 的基本原理
- JUnit 的应用

本章将学习 JUnit 的基本工作原理，以及应用 JUnit 进行单元测试。JUnit 测试框架是一个已经被多数 Java 程序员采用和实证的、优秀的测试框架。开发人员只需要按照 JUnit 的约定编写测试代码，就可以进行单元测试。本章中的重点在于如何使用 MyEclipse 集成开发平台下的 JUnit 进行测试用例(TestCase)和测试套件(TestSuite)的编写与应用。

8.1 JUnit 的基本原理

8.1.1 核心知识

JUnit 是由 Erich Gamma 和 Kent Beck 编写的一个回归测试框架(Regression Testing Framework)，供 Java 开发人员编写单元测试之用。JUnit 测试需要详细掌握待测程序的内部结构，所以属于白盒测试框架。

1. JUnit 核心类

在 JUnit 测试框架中包含 4 个核心类，分别是 TestCase、TestSuite、TestRunner 和 TestResult。

(1) TestCase(测试用例)。TestCase 中包含很多以 test 开头的方法，用来测试被测类中的 public 类型的方法。通过比较方法的输出结果和预期结果是否相同，来判断本次测试是否成功或者失败。

(2) TestSuite (测试套件)。TestCase 并不能孤立地使用，它总是需要依附在 TestSuite 中。TestSuite 代表一个或者一组 TestCase。通过 TestSuite 把 TestCase 很好地组合在一起，形成一组测试单元，也称为测试套件。

(3) TestRunner (测试运行器)。TestRunner 是负责执行 TestSuite 的程序，负责对整个测试过程进行跟踪，显示测试的结果，并报告测试的进度等。

(4) TestResult (测试结果)。TestResult 负责收集 TestCase 执行后的结果。测试结

果通常可以分为两类：客户可以预测到的错误(Failure)和不可预测的错误(Error)。

四个核心类之间的关系如图 8.1 所示。

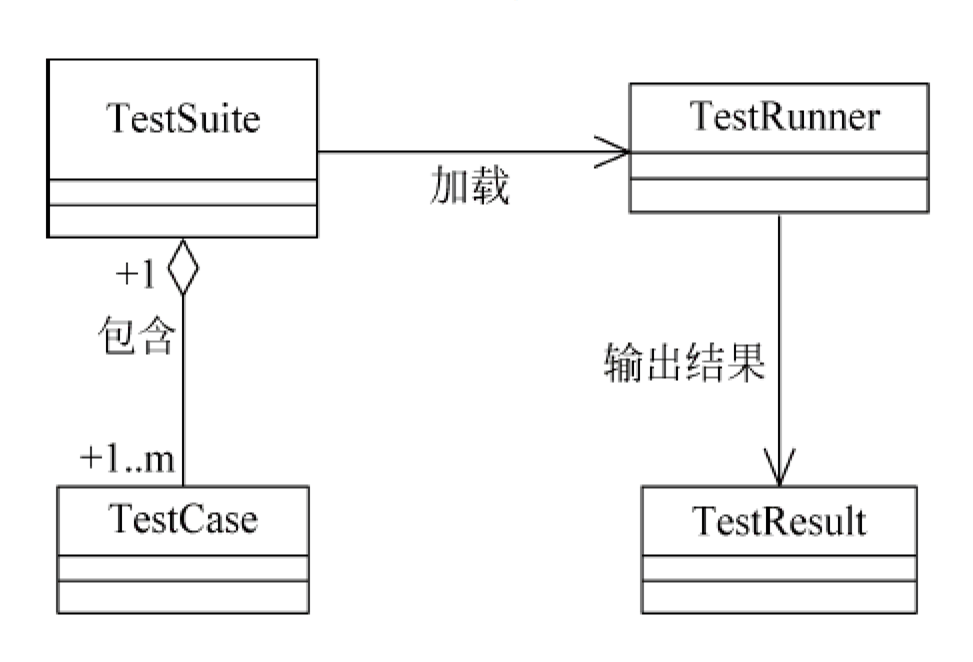


图 8.1　JUnit 核心类图

2. JUnit 中的断言

在 JUnit 测试框架中，使用断言方法来实现单元测试。断言方法以 assert 为前缀，返回一个布尔类型的值，如果返回值为 true 的话，代表测试通过；否则说明该测试单元中存在 Bug(错误)。在运行测试用例后，TestRunner 会报告哪些断言通过，哪些断言没有通过，从而快速地定位错误，而传统的测试方法都是借助于输出语句 System.out.println()等语句将信息打印到控制台后，由开发人员对输出信息进行比对后，得到测试结果。

3. 断言方法的功能描述

JUnit 中的 assert 方法全部放在 Assert 类中，JUnit 类中常用的 assert 方法分类如下。

(1) assertTrue/False([String message,]boolean condition)

assertTrue 判断为真，如果第二个参数为 false 时，输出第一个参数的内容。

assertFalse 判断为假，如果第二个参数为 true 时，输出第一个参数的内容。

例如：

```
assertTrue("结果为假",false);
```

测试结果：

```
java.lang.AssertionError: 结果为假
```

(2) assertEquals(Object expected,Object actual)

判断第一个参数和第二个参数是否相等，如果不相等时，提示错误信息。第一个参数是期望对象；第二个参数是实际对象。

例如：

```
assertEquals(3,1 + 1);
```

测试结果：

```
java.lang.AssertionError: expected:<3> but was:<2>
```

(3) assertNotNull/Null([String message,]Object obj)

assertNotNull 判断对象非空。当第二个参数为 Null 时，输出第一个参数的值。

assertNull 判断对象为空。当第二个参数不为 Null 时，输出第一个参数的值。

例如：

```
assertNotNull("对象为空",null);
```

测试结果：

```
java.lang.AssertionError: 对象为空
```

(4) assertSame/NotSame([String message,]Object expected,Object actual)

assertSame 判断第二个参数和第三个参数为同一个对象。当不是同一个对象时,输出第一个参数的值。

assertNotSame 判断第二个参数和第三个参数不是同一个对象。当是同一个对象时,输出第一个参数的值。

例如:

```
assertSame("两个对象不相等","admin","ADMIN");
```

测试结果:

```
java.lang.AssertionError: 两个对象不相等 expected same:<admin> was not:<ADMIN>
```

8.1.2 能力目标

掌握 JUint 核心类之间的关联关系,学会使用断言方法进行期望对象和实际对象的比较。

8.1.3 任务驱动

1. 任务的主要内容

(1) 创建一个待测试程序计算器类,类名为 Calculator。
(2) 定义一个属性,保存计算结果。
(3) 创建方法 add(),实现两个整数的加法运算。
(4) 创建方法 substract(),实现两个整数的减法运算。
(5) 创建方法 multiply(),实现两个整数的乘法运算。
(6) 创建方法 divide(),实现两个整数的除法运算。
(7) 创建方法 clear(),实现结果清零。

2. 任务的代码模板

将下列 Calculator.java 中的【代码】替换为程序代码。

```
Calculator.java
public class Calculator {
    【代码 1】//声明一个私有的整形属性,名字是 result
    public int add(int n,int m){
        【代码 2】//实现加法运算,把结果赋值给 result 并返回
    }
    public int substract(int n,int m){
        【代码 3】//实现减法运算,把结果赋值给 result 并返回
    }
    public int multiply(int n,int m){
        【代码 4】//实现乘法运算,把结果赋值给 result 并返回
    }
    public int divide(int n,int m) {
        【代码 5】///实现除法运算,把结果赋值给 result 并返回
    }
```

```
    public void clear(){
        【代码 6】//对 result 重新赋值为 0
    }
}
```

3. 任务小结或知识扩展

计算器类计算出来的结果可能带有小数，这时候可以考虑把 result 的类型换成 double，对应的方法返回类型以及参数也需要发生改变。

4. 代码模板的参考答案

【代码 1】：private int result;

【代码 2】：result = m + n;
　　　　　return result;

【代码 3】：result = n - m;
　　　　　return result;

【代码 4】：result = n * m;
　　　　　return result;

【代码 5】：result = n / m;
　　　　　return result;

【代码 6】：result = 0;

8.1.4 实践环节

1. 编写一个测试类 Test.java，包含 main 方法。
2. 在 main 方法中，创建一个计算器对象。
3. 通过对象名调用 add()方法，并对结果进行控制台输出，观察计算结果。
4. 通过对象名调用 clear()方法，对结果进行清零。
5. 通过对象名调用 substract ()方法，并对结果进行控制台输出，观察计算结果。
6. 通过对象名调用 clear()方法，对结果进行清零。
7. 通过对象名调用 multiply ()方法，并对结果进行控制台输出，观察计算结果。
8. 通过对象名调用 clear()方法，对结果进行清零。
9. 通过对象名调用 divide ()方法，并对结果进行控制台输出，观察计算结果。

8.2 JUnit 的应用

8.2.1 核心知识

1. 添加 JUnit 测试框架

MyEclipse 是 Java 语言的集成开发平台，内部已经包含了 JUnit 测试框架，只需要在自己的 Java 项目中添加对应的 jar 包，就可以使用。

(1) 创建一个 Java Project，名字为 JUnitProject。操作步骤如下。

选择 File→New→Java Project 命令，在弹出的窗口的 Project name 文本框中输入

JUnitProject，单击 Finish 按钮完成，如图 8.2 所示。

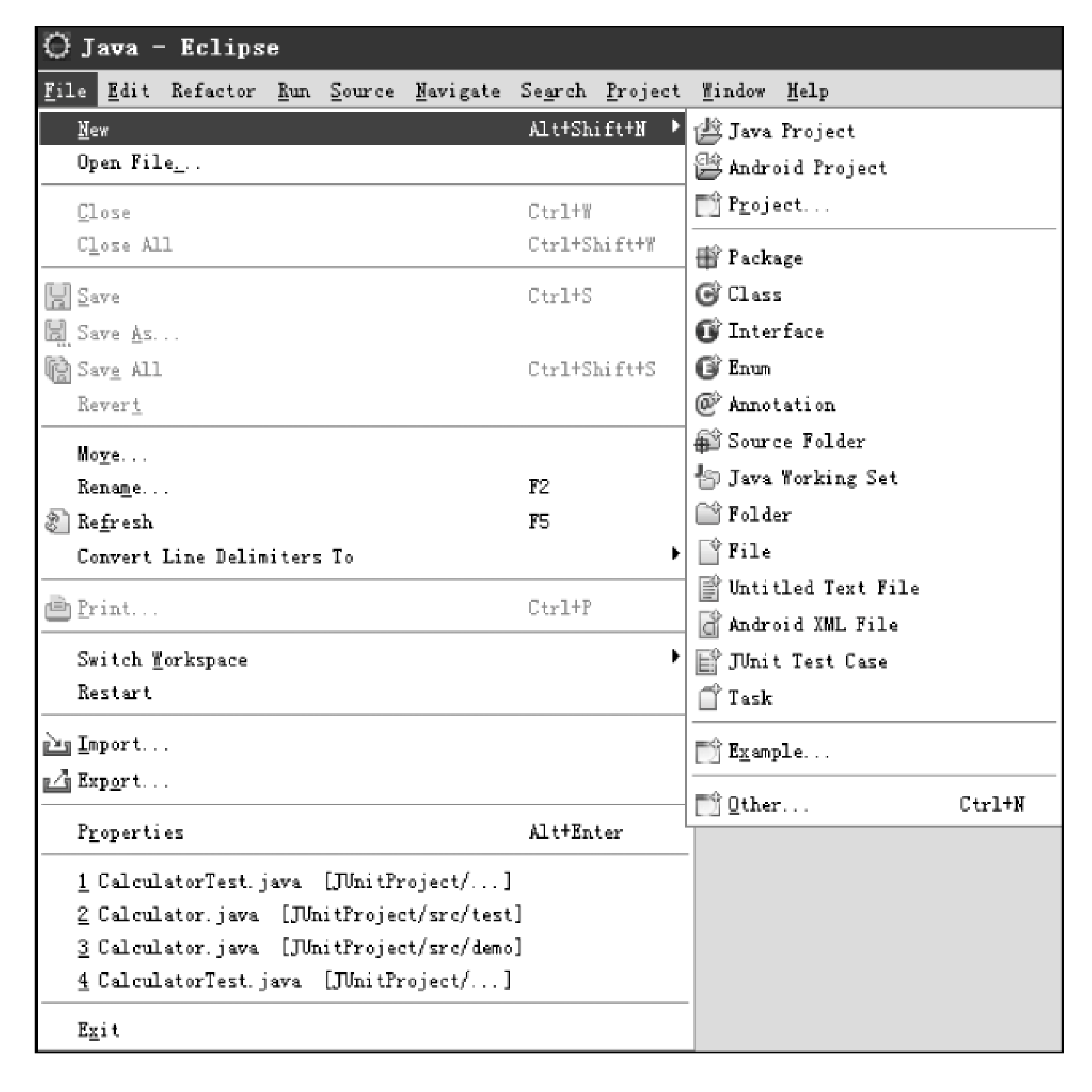


图 8.2 创建 Java Project

（2）为项目添加 JUnit 测试框架。选中项目名称右击，在弹出的菜单中选择 Properties 命令，打开如图 8.3 所示窗口。

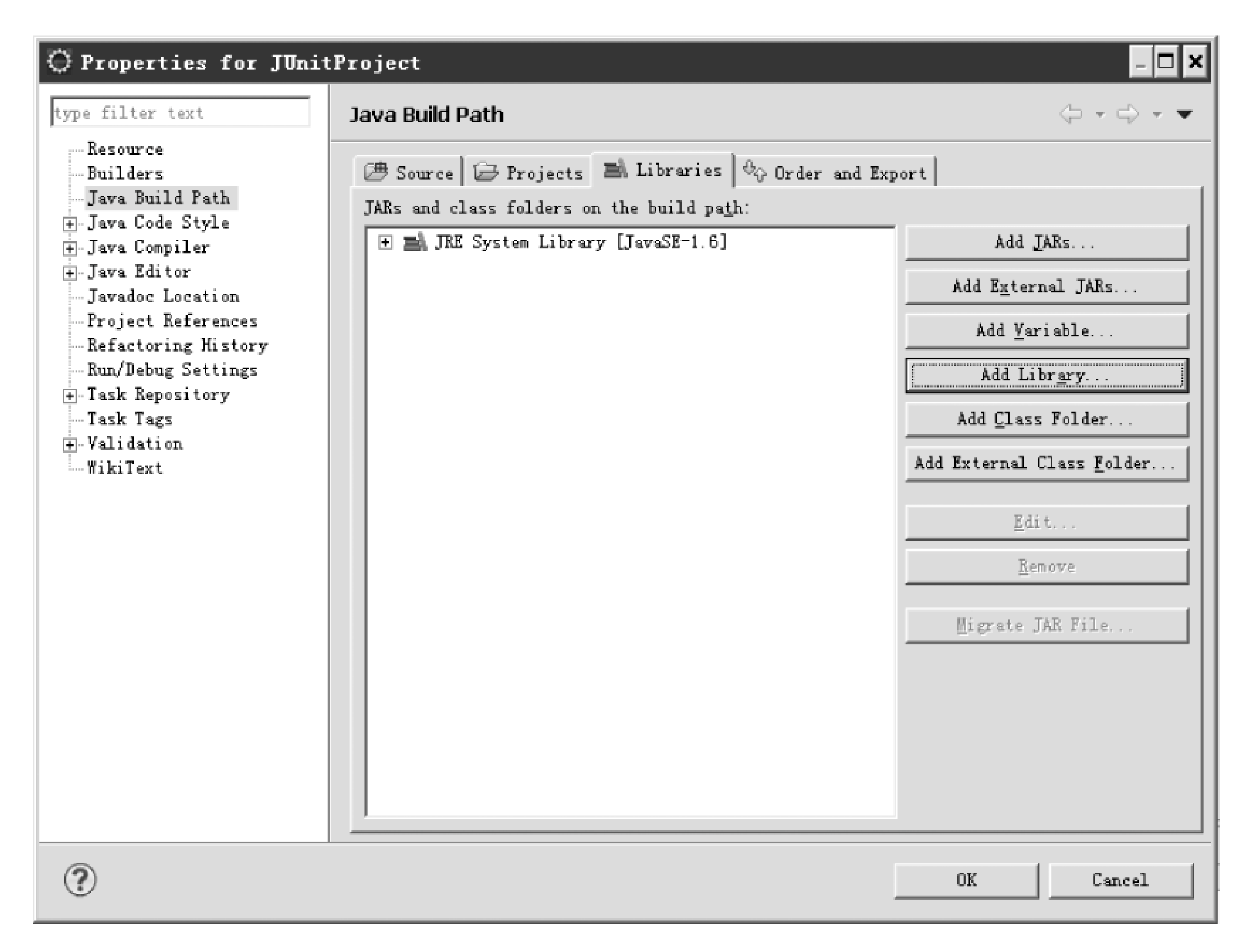


图 8.3 Java Project 添加第三方库

在左侧栏中选择 Java Build Path 选项，再在右侧选择 Libraries 选项卡，单击 Add Library 按钮，在弹出的 Add Library 窗口中选择 JUnit 选项，再单击 Next 按钮，然后在下拉列表中选择 JUnit 4 选项，最后单击 Finish 按钮，完成 JUnit 测试框架的添加，如图 8.4 所示。

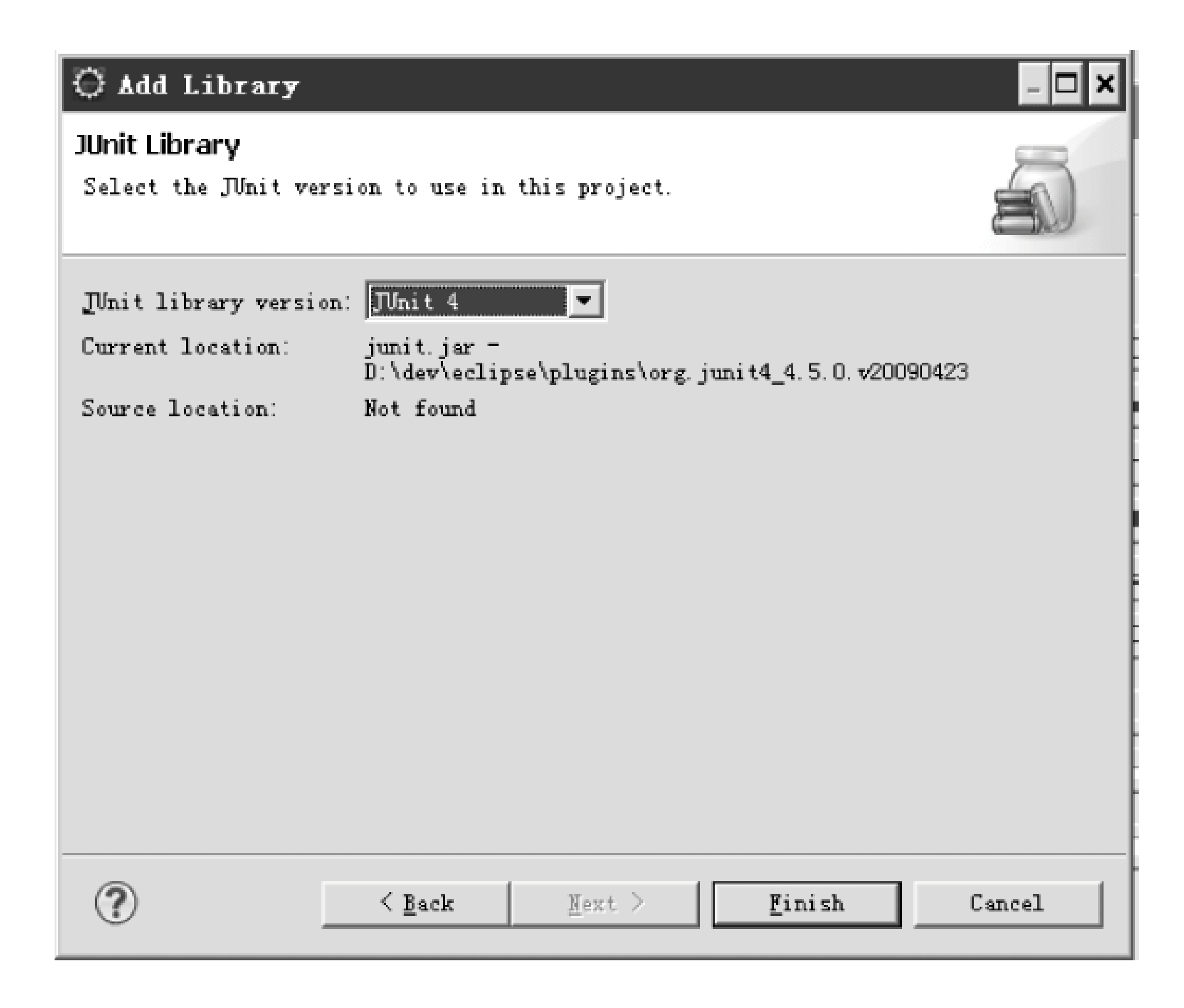


图 8.4　添加 JUnit 测试框架

2. 创建 TestCase

对上节创建的计算器类编写 TestCase。操作步骤如下。

(1) 选择待测试的类 Calculator 右击，在弹出的菜单中选择 New→JUnit Test Case 命令，如图 8.5 所示，然后单击 Next 按钮。

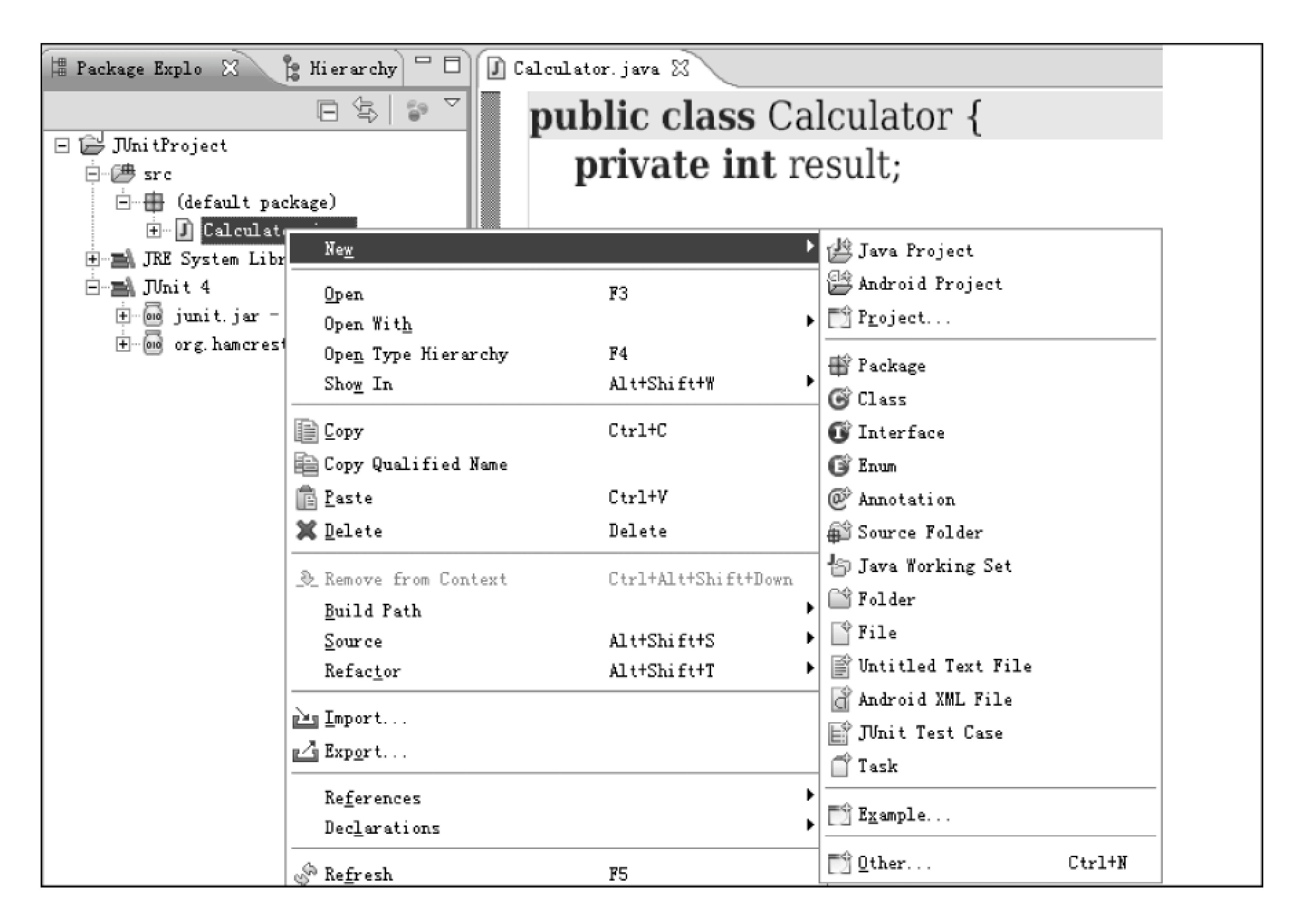


图 8.5　创建计算器类的 TestCase

(2) 选择待测试的方法。这里把 Calculator 类的四个方法都选上,单击 Finish 按钮,如图 8.6 所示。

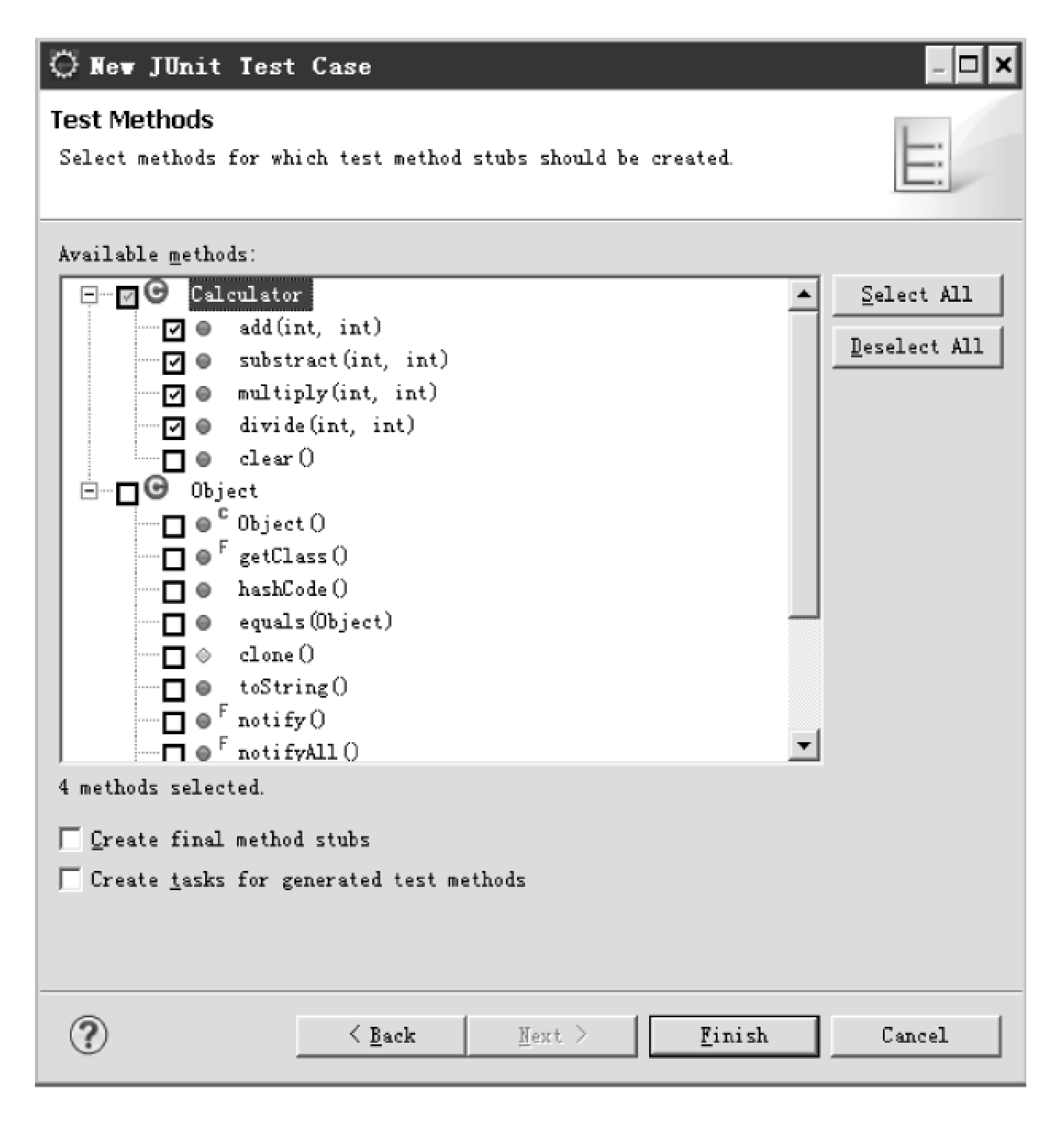


图 8.6 选择待测试的方法

框架自动生成对 Calculator 的测试类 CalculatorTest,代码如下。

```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;
import junit.framework.TestCase;
public class CalculatorTest extends TestCase{
    @Before
    public void setUp() throws Exception {
    }
    @Test
    public void testAdd() {
        fail("Not yet implemented");
    }
    @Test
    public void testSubstract() {
        fail("Not yet implemented");
    }
    @Test
    public void testMultiply() {
        fail("Not yet implemented");
    }
    @Test
    public void testDivide() {
```

```
            fail("Not yet implemented");
        }
    }
```

代码说明：

第一行 import static org.junit.Assert.*；表示静态引入断言方法。例如 assertEquals 是 Assert 类中的一个静态方法，一般的使用方式是 Assert.assertEquals()，但是使用了静态包含后，前面的类名就可以省略了，使用起来更加的方便。

@Before 和@Test 都代表“固定代码段”。@Before 说明 setUp()方法在任何一个测试方法执行之前都必须先执行 setUp()方法中的代码。这里可以调用计算器清零的方法，因为每次在进行新的计算之前，都需要清零操作。

@Test 下的所有方法都是以 test 为前缀，是用来对每个 Calculator 类中的方法进行测试，返回类型为 void。

3. 编写断言

在本例中主要采用 assertEquals()断言方法进行单元测试。代码如下。

```
public class CalculatorTest extends TestCase {
    Calculator calculator = new Calculator();
    @Before
    public void setUp() throws Exception {
        calculator.clear();
    }
    @Test
    public void testAdd() {
        assertEquals(3, calculator.add(1, 2));
    }
    @Test
    public void testSubstract() {
        assertEquals(1, calculator.substract(1, 2));        //预期结果应该为 -1
    }
    @Test
    public void testMultiply() {
        assertEquals(2, calculator.multiply(1, 2));
    }
    @Test
    public void testDivide() {
        assertEquals(0, calculator.add(1, 0));              //除以 0,出现异常
    }
}
```

代码说明：

首先，在测试用例 CalculatorTest 类中创建一个被测类 Calculator 的对象：Calculator calculator=new Calculator()。在每个测试之前执行 setUp()。计算器在计算前需要清零操作：calculator.clear()。

接着，是在各个测试方法中使用 assertEquals()断言方法判断计算器计算出来的结果和预期结果是否相等。如果相等，本方法测试通过。

4. 运行 TestCase

测试用例编写好之后，可以运行测试用例。在菜单栏选择 Run→Run As→Junit Test 命令，或者单击绿色向右箭头图标同样可以运行。

5. 观察测试结果

运行测试用例后，会得到测试结果，如图 8.7 所示。

运行结果说明运行了 4 个方法，成功通过测试的方法为 testAdd()、testMultiply()，错误的方法为 testSubstract()，出现异常的方法为 testDivide()。

单击错误的方法后，Failure Trace 文本区域就会出现错误信息，如图 8.8 所示。

说明：assertEquals(1，calculator.substract(1，2))的断言方法，期望值是 1，但是实际值是－1，所以测试失败。通过错误信息可以很快定位到测试有问题的代码处，进行改进。

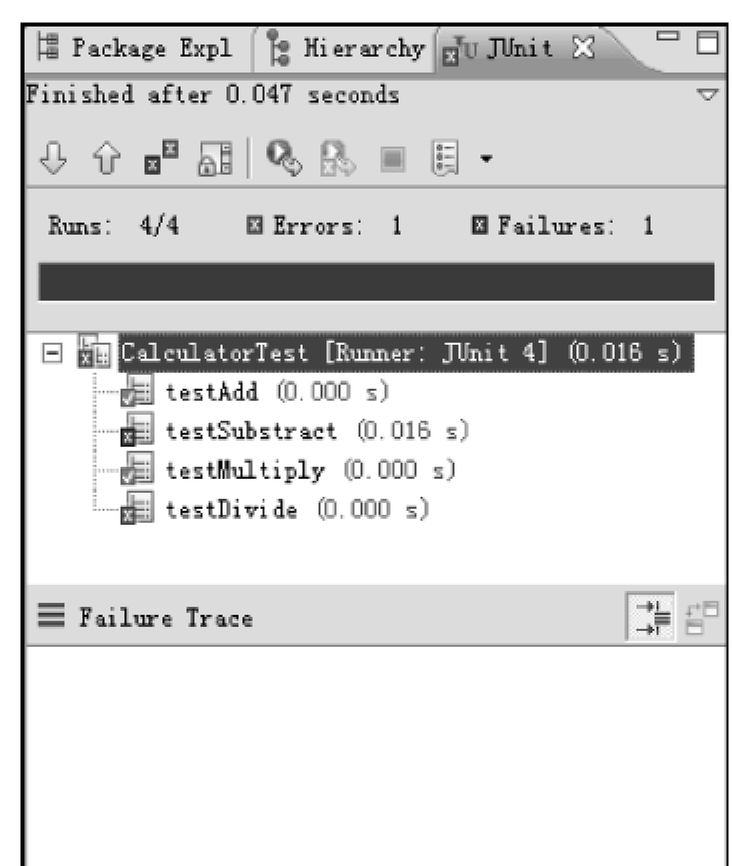


图 8.7　测试结果

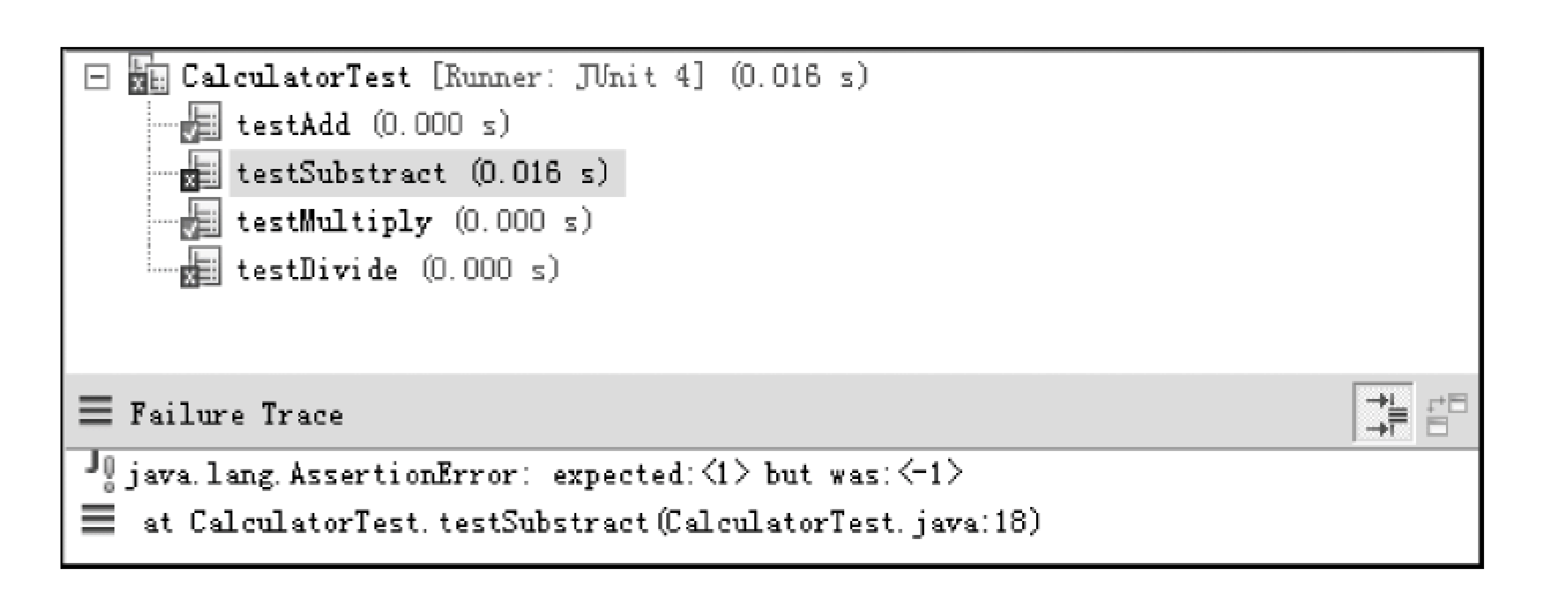


图 8.8　错误信息

单击异常的方法后，Failure Trace 文本区域就会出现异常的信息，如图 8.9 所示。

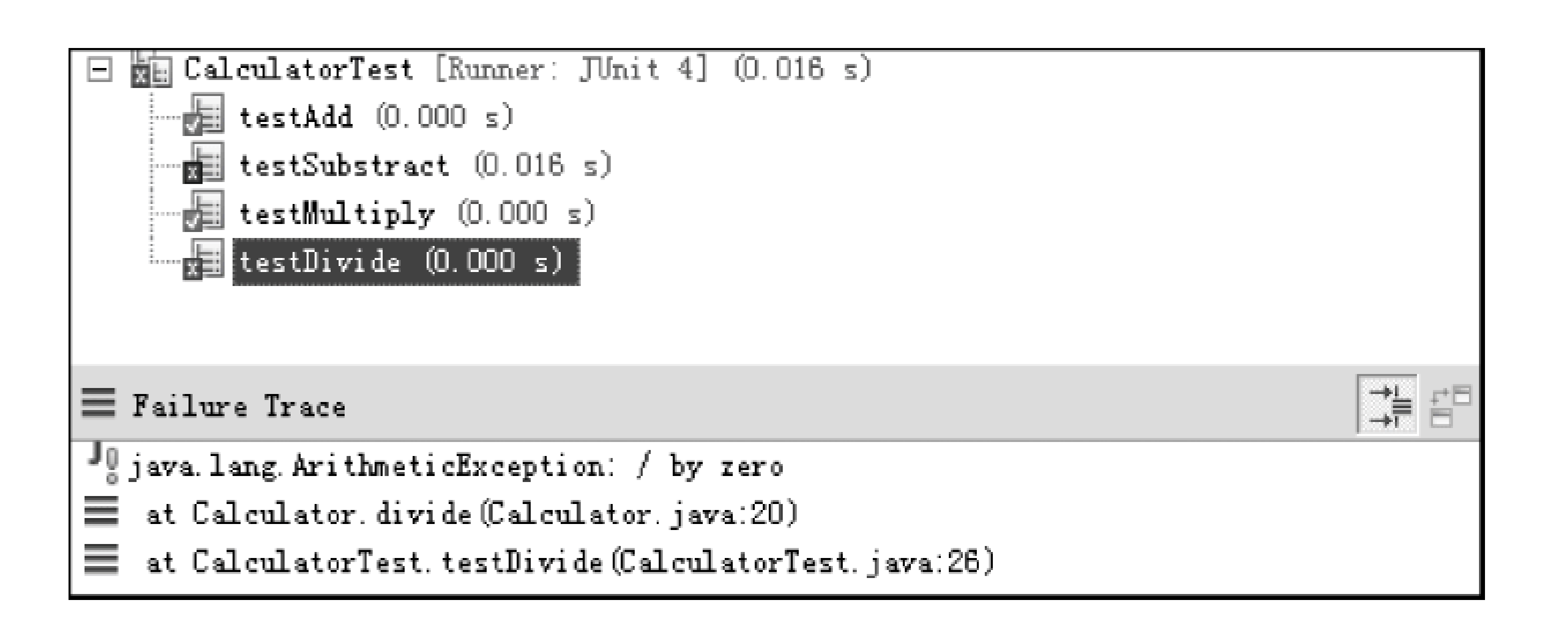


图 8.9　特定错误信息

说明：assertEquals(0，calculator.add(1，0))的断言方法出现算数异常，所以测试失败。通过异常信息可以很快定位到测试有问题的代码处，进行改进。

6. 创建并运行 TestSuite

TestCase 需要有 TestSuite 才能运行，如果编程人员没有提供 TestSuite 时，TestRunner 会自动创建一个测试套件。

TestSuite 的作用主要有两个：对多个 TestCase 归为一组进行测试；对单个 TestCase 中的方法进行单独测试。

(1) 对多个 TestCase 测试

上例中针对计算器类创建了一个 TestCase，类名为 CalculatorTest。接着按照相同的方式创建另一个类 SmallestNumber，用来获得整型数组中最小的数，同时创建出该类的 TestCase，类名为 SmallestNumberTest。代码如下。

```
SmallestNumber.java
public class SmallestNumber {
    public int getSmallest(int nums[]){
        int min = nums[0];
        for(int i = 1;i < nums.length;i++){
            if(min > nums[i]){
                min = nums[i];
            }
        }
        return min;
    }
}
```

代码说明：

在 getSmallest()方法中传入一个整型数组 nums。把数组中第一个元素赋值给局部变量 min，通过循环比较，得到该数组中最小的整数并返回。

```
SmallestNumberTest.java
import static org.junit.Assert.*;
import junit.framework.TestCase;
import org.junit.Test;
public class SmallestNumberTest extends TestCase{
    SmallestNumber sn = new SmallestNumber();
    @Test
    public void testGetSmallest() {
        int nums[] = {20,14,1,100};
        assertEquals(1, sn.getSmallest(nums));
    }
}
```

在 SmallestNumberTest 类中有个测试方法 testGetSmallest()，通过断言 assertEquals()判断测试数组 nums 中的最小整数是否是 1。如果是 1，代表该 TestCase 测试通过。

测试用例 SmallestNumberTest 编写完成后，接着创建 TestSuite。方法为：选择 File→New→Other 命令，在弹出的 New 对话框的列表框中选择 Java/JUnit/JUnit Test Suite 选项，打开 New Junit Test Suite 对话框，如图 8.10 所示。

TestSuite 名称为 AllTests。Package 一栏中没有填写，表示 TestSuite 放在默认包中保存(系统提示警告，不推荐使用，以后可以把 TestSuite 放在指定的测试套件包中)。默认情况下系统自动把两个 TestCase 加入到该 TestSuite 中，单击 Finish 按钮。自动生成如下

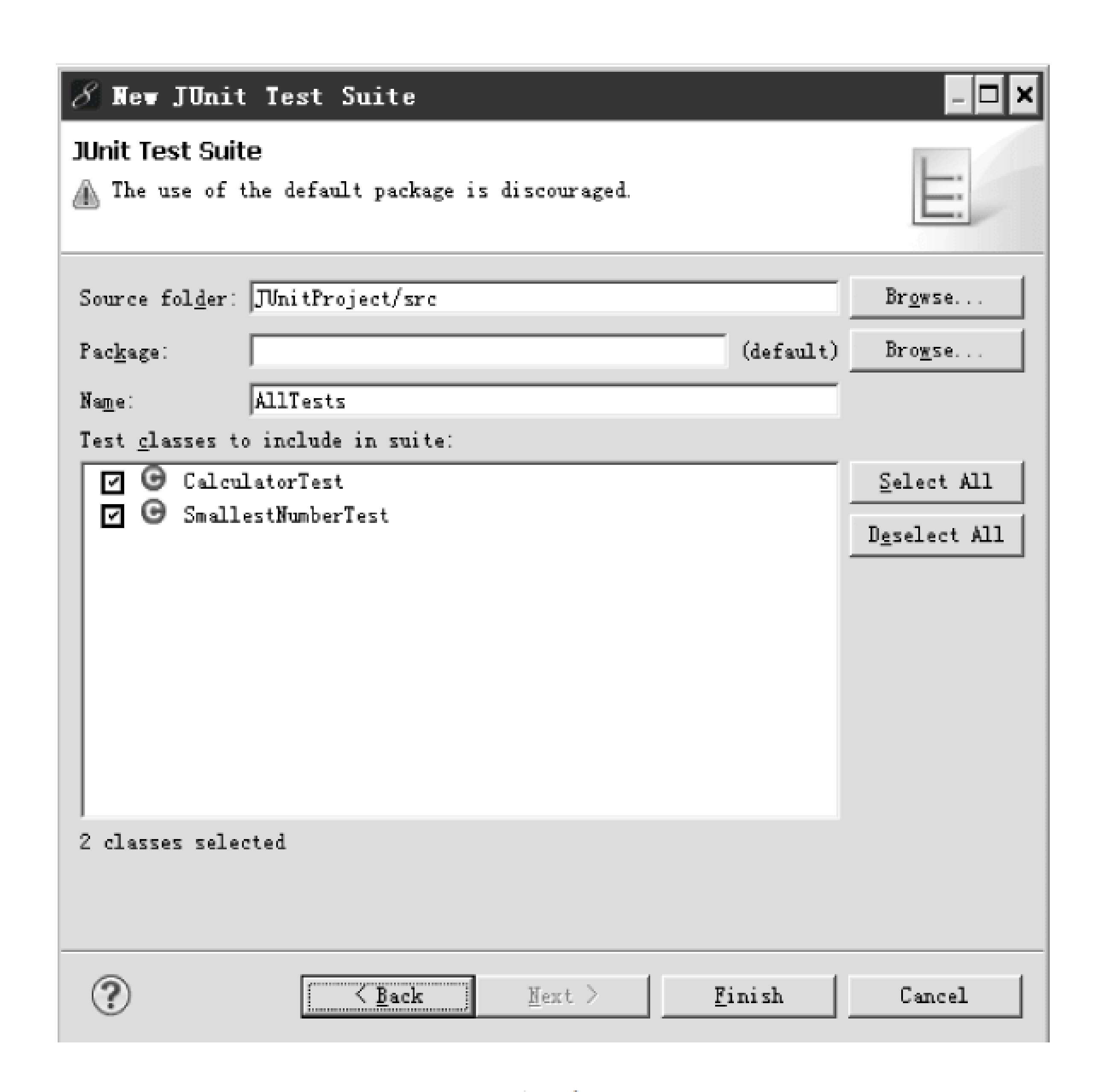


图 8.10　创建 TestSuite

代码。

```
import junit.framework.Test;
import junit.framework.TestSuite;
public class AllTests {
    public static Test suite() {                              //1 行
        TestSuite suite = new TestSuite("Test for default package");     //2 行
        // $ JUnit - BEGIN $
        suite.addTestSuite(CalculatorTest.class);             //3 行
        suite.addTestSuite(SmallestNumberTest.class);         //4 行
        // $ JUnit - END $
        return suite;
    }
}
```

代码说明：

1 行：静态方法，返回 TestSuite 类对象。该方法主要是构造 TestSuite 对象，然后向其中加入想要测试的方法。

2 行：构造 TestSuite 类对象。

3、4 行：向 TestSuite 对象中添加两个 TestCase 作为一组进行测试。

最后，运行该 TestSuite。方法为：选择 Run→Run As→JUnit Test 命令。运行结果如图 8.11 所示。

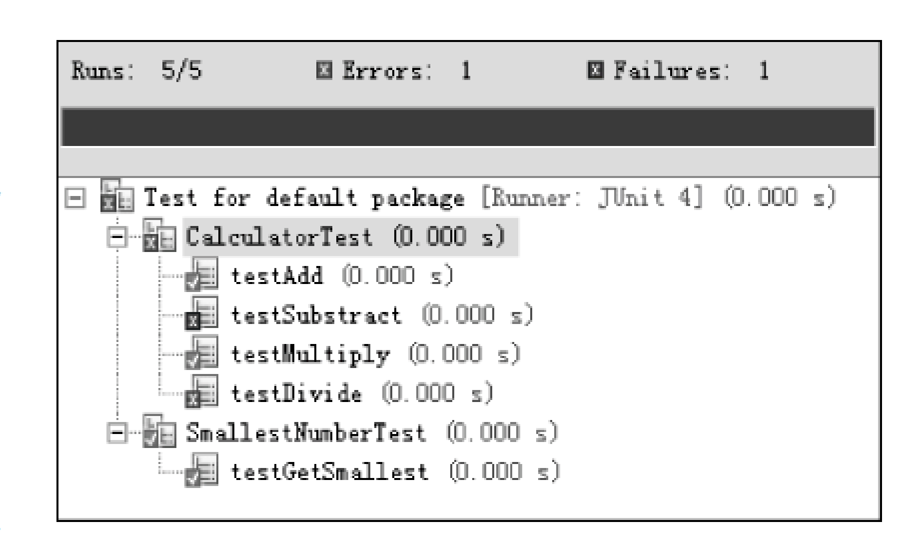


图 8.11　TestSuite 运行结果

（2）对单个 TestCase 中的方法测试

之前每次运行测试类时，该测试类的所有方法全部被测试一遍，如果想单独测试某个方法比较麻

烦，这时可以利用测试套件来解决这个问题。

下面先向 CalculatorTest 中添加一个构造方法，目的是调用父类的构造方法，决定调用类中的哪一个测试方法。代码如下。

```
public CalculatorTest(String name){
    super(name);
}
```

在 AllTests 类中的代码修改如下：

```
public class AllTests {
    public static Test suite() {
        TestSuite suite = new TestSuite("Test for default package");
        //$JUnit-BEGIN$
        suite.addTest(new CalculatorTest("testDivide"));    //1 行
        //$JUnit-END$
        return suite;
    }
}
```

代码说明：

1 行：向 TestSuite 对象中添加一个要测试的方法。new CalculatorTest("testDivide")表示测试 CalculatorTest 类的 testDivide 方法。运行后，测试结果如图 8.12 所示。

8.2.2 能力目标

掌握 JUnit 测试框架的添加过程；掌握 TestCase 和 TestSuite 的编写；学会使用断言进行单元测试；学会观察测试结果，找出错误的情况以及定位错误。

图 8.12 testDivide 方法测试结果

8.2.3 任务驱动

1. 任务的主要内容

(1) 创建一个待测试程序银行账户类，类名为 Account。

(2) 定义两个属性。分别是 String 类型的“账号”和 double 类型的“账户金额”。

(3) 创建构造方法 Account()，实现账户的初始化功能。

(4) 创建方法 saveMoney()，实现向账号存钱功能。

(5) 创建方法 drawMoney()，实现取钱功能。

(6) 创建方法 getBalance()，实现获得余额功能。

(7) 创建 TestCase，名称为 AccountTest。

(8) 在 AccoutTest 类中对上述定义的方法进行单元测试。

(9) 创建 TestSuite，名称为 AccountTestSuite。

(10) 在 AccountTestSuite 类中对上述定义的方法进行单元测试。

2. 任务的代码模板

将下列类中的【代码】替换为程序代码。

```
Account.java
public class Account {
    【代码 1】                //声明一个私有的字符串属性,名字是 accountId
    private double money;
    public Account(String accountId){
        this.accountId = accountId;
        【代码 2】            //账户金额初始化为 0
    }
    public void saveMoney(double m)
    {
        【代码 3】            //向账户金额中存钱,金额为 m
    }
    public void drawMoney(double m)
    {
        money = money - m;
    }
    public double getBalance() {
        【代码 4】            //返回账户金额
    }
}
AccountTest.java
public class AccountTest extends TestCase{
    【代码 5】                //创建一个银行账户对象,账号为"A001"
    @Before
    public void setUp() throws Exception {
    }
    @Test
    public void testSaveMoney() {
        【代码 6】            //存入 1000
        【代码 7】            //使用断言判断期望余额结果和实际计算余额结果是否相等
    }
    @Test
    public void testDrawMoney() {
        acc.saveMoney(1000);
        【代码 8】            //取出 400
        【代码 9】            //使用断言判断期望余额结果和实际计算余额结果是否相等
    }
    @Test
    public void testGetBalance() {
    }
}
AccountTestSuite.java
public class AccountTestSuite {
    public static Test suite() {
        TestSuite suite = new TestSuite("Test for default package");
【代码 10】                   //把 AccountTest 添加到测试套件中
        return suite;
    }
}
```

3. 任务小结或知识扩展

需要注意的是，double 类型的属性，在赋值的过程中，如果是整数的话，会自动在末尾补上“.0”。在测试 testSaveMoney()和 testDrawMoney()时，账户金额都是初始值 0。所以【代码 7】中的第一个参数值为 1000.0 时，该断言为真；【代码 9】中的第一个参数值为 600.0 时，该断言为真。

4. 代码模板的参考答案

【代码 1】：private String accountId;

【代码 2】：money = 0;

【代码 3】：money = money + m;

【代码 4】：return money;

【代码 5】：Account acc = new Account("A001");

【代码 6】：acc.saveMoney(1000);

【代码 7】：assertEquals(1000.0, acc.getBalance());

【代码 8】：acc.drawMoney(400);

【代码 9】：assertEquals(600.0, acc.getBalance());

【代码 10】：suite.addTestSuite(AccountTest.class);

8.2.4 实践环节

1. 编写银行账户类 Account 以及存款、取款、查询余额等方法。
2. 编写 TestCase，名称为 AccountTest。对 Account 中的方法进行单元测试。
3. 运行 AccountTest，观察测试结果。
4. 针对测试结果，调整 Account 类，使其中的方法编写正确。
5. 编写 TestSuite，名称为 AccountTestSuite。只对 AccountTest 中的 testDrawMoney()进行测试。

8.3 小　结

- 在 JUnit 测试框架中包含 4 个核心类，分别是 TestCase、TestSuite、TestRunner 和 TestResult。TestCase 用来测试被测类中的 public 类型的方法。TestSuite 代表一个或者一组 TestCase。TestRunner 负责执行 TestSuite 的程序。TestResult 负责收集 TestCase 执行后的结果。
- 在 JUnit 测试框架中，使用断言方法来实现单元测试。常用的断言方法有：assertTrue()、assertEquals()、assertNotNull()、assertSame()等。

习　题　8

1. TestCase 和 TestSuite 有什么区别?
2. 断言方法 assertEquals()是如何进行测试的?

3. 下面哪个是单元测试的测试工具？(　　)

A. JUnit　　B. WinRunner　　C. QTP　　D. Tomcat

4. JUnit 中测试用例必须继承的父类是(　　)。

A. TestCase　　B. TestSuite　　C. Test　　D. TestDemo

5. 按题目要求编写相关类，并使用 JUnit 对计算周长和面积的方法进行单元测试。

梯形 Lader 类具有类型为 double 的上底、下底、高、面积属性，具有返回面积的功能，包括一个构造方法对上底、下底、高进行初始化。圆形 Circle 类具有类型为 double 的半径、周长和面积属性，具有返回周长、面积的功能，包括一个构造方法对半径进行初始化。

附　　录

参 考 答 案

习题 1 答案

1. 什么是软件测试？简述其目的与原则。

答：软件测试是通过人工或者自动手段来运行或测试某个系统的过程，从而验证软件是否能达成期望功能，它是验证软件期望功能的唯一有效方法，也是保证软件产品质量的唯一途径。软件测试并非是简单的“挑错”，而是贯穿于软件生产过程的始终，一套完善的质量体系。

基于不同的立场，存在着两种完全不同的测试目的。从用户的角度出发，普遍希望通过软件测试暴露出软件中隐藏的错误和缺陷，以考虑是否可以接受该产品。而从软件开发者的角度出发，则希望测试成为表明软件产品中不存在错误的过程，验证该软件已正确地实现了用户的要求，确立用户对软件质量的信心。

在软件测试的过程中，通常应该遵循以下 7 个原则。

(1) 所有的测试都应追溯到用户需求。

(2) 应尽早地和不断地进行软件测试。测试工作进行得越早，越有利于提高软件的质量，这是预防性测试的基本原则。

(3) 在有限的时间和资源下进行完全测试找出软件所有的错误和缺陷是不可能的，软件测试不能无限进行下去，应适时终止。

(4) 测试只能证明软件存在错误而不能证明软件没有错误，测试无法显示潜在的错误和缺陷，继续进一步测试可能还会找到其他错误和缺陷。

(5) 充分关注测试中的集群现象。在测试的程序段中，若发现的错误数目多，则残存在其中的错误数目也比较多，因此应当花较多的时间和代价测试那些具有更多错误数目的程序模块。

(6) 程序员应避免检查自己的程序。

(7) 尽量避免测试的随意性。软件测试是有组织、有计划、有步骤的活动，要严格按照测试计划进行。

2. 影响软件测试的因素有哪些？

答：(1) 软件系统的目的。安全性级别越高的系统要求有更完备的软件测试。

(2) 潜在的用户数量。系统用户的数量越多，越需要更完备的软件测试。

(3) 信息的价值。信息的经济性越高，价值越高，越需要更完备的软件测试。

(4) 开发机构。一个没有标准和缺少经验的开发机构很可能开发出充满错误的系统。

(5) 测试的时机。测试量会随时间的推移发生改变，应该针对合适的目标进行调整。

3. 简述软件测试终止的标准。

答：(1)基于“测试阶段”的原则。可以分别对单元测试、集成测试和系统测试制定详细

的测试结束点。

(2) 基于“测试用例”的原则。测试设计人员设计测试用例，并请项目组成员参与评审，测试用例一旦评审通过，后面测试时，就可以作为测试结束的一个参考标准。

(3) 基于“缺陷收敛趋势”的原则。随着测试工作量的不断增加，缺陷数量应该呈下降趋势。

(4) 基于“缺陷修复率”的原则。根据软件缺陷的严重程度实施不同的缺陷修复标准。

(5) 基于“验收测试”的原则。要求通过用户的测试验收。

4. 软件测试人员应该具备哪些职业素质？

答：(1)技术能力

(2) 具有一定的编程经验

(3) 沟通能力

(4) 要有严谨、敢于承担责任、稳重的做事风格

(5) 具有怀疑与破坏的精神

(6) 善于自我总结、自我督促

(7) 团队合作能力

5. B　　6. B　　7. D　　8. B　　9. A　　10. D

习题 2 答案

1. 什么是静态测试、动态测试、黑盒测试、白盒测试？

答：静态测试：不实际运行软件，发挥人的逻辑思维优势，主要对软件代码的逻辑、程序结构等方面进行评估。主要包括需求评审、设计评审、代码走查、代码检查等。

动态测试：实际运行软件，发现系统中存在的错误。单元测试、集成测试、系统测试、确认测试、验收测试、回归测试等阶段都包含有动态测试的内容。

黑盒测试：也称为功能测试，它是通过测试来检测每个功能是否都能正常使用。着眼于程序外部结构，不考虑内部逻辑结构如何。主要针对软件界面和软件功能进行测试。

白盒测试：也称为结构测试或逻辑驱动测试，它是按照程序内部的结构测试程序，通过测试来检测产品内部动作是否按照设计规格说明书的规定正常进行，检验程序中的每条通路是否都能按预定要求正确工作。这一方法是把测试对象看作一个打开的盒子，测试人员依据程序内部逻辑结构相关信息，设计或选择测试用例，对程序所有逻辑路径进行测试，通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致。

2. 如果开发时间紧迫，是否可以跳过单元测试而直接进行集成测试？试说明原因。

答：不可以。单元测试又称为模块测试，主要检查每个程序单元是否正确实现详细设计说明中的模块功能。单元测试阶段通常可以排除系统 60%以上的错误，如果没有进行单元测试，将造成集成测试无法顺利进行。单元测试阶段的错误，会在集成测试阶段产生集群效应。最终造成整个软件测试的失败。

3. 什么是驱动模块和桩模块？

答：驱动模块：用来调用被测模块。如 Java 语言，在类中写好一个方法后，为了验证这个方法的有效性，可以在主方法 main 中去调用该被测方法。

桩模块：用来支撑被测模块。如在网上银行系统中，想要测试转账模块是否好用，转账

模块需要用户首先正确登录后才能使用,这时候登录模块就是转账模块的桩模块。

4. 集成测试常用的集成策略有哪些？分别说明。

答：集成测试的策略主要分为两种：非增量式集成测试和增量式集成测试。

非增量式集成测试：是采用一步到位的方法来构造测试。对所有模块进行单元测试后,按照程序结构图将各模块连接起来,把连接后的程序当作一个整体进行测试。

增量式集成测试是逐次将未曾集成测试的模块和已经集成测试的模块(或子系统)结合成程序包,再将这些模块集成为较大的系统,在集成的过程中边连接边测试,以发现连接过程中产生的问题。按照不同的实施次序,增量式集成测试又可以分为三种不同的方法：自顶向下增量式集成测试、自底向上增量式集成测试和混合增量式集成测试。

5. 软件测试流程包括几个步骤？

答：软件测试流程包括5个步骤：测试需求分析、测试计划、测试用例设计、测试执行和测试评估。

测试需求分析：主要目的是对需求设计的理解和用户需求的理解,把不直观的需求转变为直观的需求,把不明确的需求转变为明确的需求,把不能度量的需求转变为可度量的需求,从而进行测试设计。

测试计划：描述了要进行的测试活动的范围、方法、资源和进度的文档。它确定测试项、被测特性、测试任务、谁执行任务、各种可能的风险。

测试用例设计：测试用例是为特定的目的而设计的一组测试输入、执行条件和预期的结果的集合。测试用例是执行的最小实体。简单地说,测试用例就是设计一个场景,使软件程序在这种场景下,必须能够正常运行并且达到程序所设计的执行结果。

测试执行：测试执行一般会经历以下三个阶段。初测期,测试主要功能和关键执行路径,排除主要障碍。细测期,依据测试计划和测试用例,逐一测试功能、性能、用户界面、兼容性、可用性等。回归测试期,复查已知错误的纠正情况,确认未引发任何新的错误时,终结回归测试。

测试评估：测试报告的目的是总结当前的软件测试工作,对被测试软件的版本质量作出评估,给产品能否发布一个参考值。

6. 分析比较面向对象的软件测试与传统的软件测试的异同。

答：传统的单元测试的对象是软件设计的最小单位——模块。当考虑面向对象软件时,单元的概念发生了变化,此时最小的可测试单位是封装的类或对象,而不再是个体的模块。传统单元测试主要关注模块的算法实现和模块接口间数据的传递,而面向对象的单元测试主要考察封装在一个类中的方法和类的状态行为。

面向对象软件没有层次的控制结构,因此传统的自顶向下和自底向上集成策略就不再适合,它主要有以下两种集成策略：基于类间协作关系的横向测试；基于类间继承关系的纵向测试。系统测试一般不考虑内部结构和中间结果,因此面向对象软件系统测试与传统的系统测试差别不大。

面向对象软件测试的整体目标和传统软件测试的目标是一致的,即以最小的工作量发现尽可能多的错误,但是面向对象测试的策略和战术有很大不同。测试的视角扩大到包括复审分析和设计模型,此外,测试的焦点从过程构件(模块)移向了类。

7. C　8. B　9. C　10. A　11. B　12. A　13. B　14. D　15. D

16. C　17. B　18. D　19. B　20. A　21. A　22. D　23. D　24. D
25. C　26. C

习题 3 答案

1. 简述黑盒测试方法的特点。

答：黑盒测试又称为功能测试或数据驱动测试，主要针对软件界面、软件功能、外部数据库访问以及软件初始化等方面进行测试。

黑盒测试的主要优点如下。

(1) 从产品功能角度测试可以最大限度地满足用户的需求。

(2) 相同动作可重复执行，最枯燥的部分可由机器完成。

(3) 依据测试用例有针对性地寻找问题，定位更为准确，容易生成测试数据。

(4) 将测试直接和程序/系统要完成的操作相关联。

黑盒测试的主要缺点如下。

(1) 代码得不到测试。

(2) 如果规格说明设计有误，很难发现。

(3) 测试不能充分地进行。

(4) 结果的准确性取决于测试用例的设计。

2. 标准等价类划分法和健壮等价类划分法的区别是什么？

答：标准等价类不考虑无效数据值，测试用例使用每个等价类中的一个值；通常，标准等价类测试用例的数量和最大等价类中元素的数目相等。

健壮等价类对有效输入，测试用例从每个有效等价类中取一个值；对无效输入，一个测试用例有一个无效值，其他值均取有效值；通常，规格说明往往没有定义无效测试用例的期望输出，因此需要定义这些测试用例的期望输出。

3. 决策表分析法为什么可以进行完备测试？

答：决策表分析法就是分析和表达多逻辑条件下执行不同操作情况的黑盒测试方法。由于它能把所有条件的组合罗列出来，并针对每种组合执行不同的动作，故可以进行完备测试。

4. 测试银行提款机上的提款功能，要求用户输入的提款金额的有效数值是 100～3000，并以 100 为最小单位(即取款金额为 100 的倍数)试用等价类划分法和边界值分析法设计测试用例。

答：(1) 等价类划分法

首先，根据提款金额的有效范围是 100～3000，可以划分三个等价区间分别是：

有效等价类　100＜＝取款金额＜＝3000；

无效等价类　0＜＝取款金额＜100；取款金额＞＝3000。

其次，根据取款金额为 100 的倍数，可以划分两个等价区间分别是：

有效等价类　取款金额是 100 的倍数；

无效等价类　取款金额不是 100 的倍数。

最后，得到等价类划分法的测试用例如下表。

取款模块价类划分法的测试用例表

测试模块	测试方法	测试区间	测试用例编号	预期输入	预期输出
取款模块	有效等价类	金额在 100～3000 并且是 100 的倍数	1	1000	取款成功
	无效等价类	0<=金额<100	2	50	取款金额最低 100
		金额>3000	3	3500	取款金额超过单笔最高金额
		金额在 100～3000 但不是 100 的倍数	4	250	请正确输入取款金额

（2）边界值分析法

由于银行提款机的操作页面设计时已经去除了取款金额为负数的情况，所以取款金额的边界分别为[0,3000]的 100 的整数倍。

取款模块标准边界值测试用例表

测试模块	测试方法	测试区间	测试用例编号	预期输入	预期输出
取款模块	边界值分析法	等于 0	1	0	请输入取款金额
		略大于 0	2	1	取款金额最低 100
		金额在 100～3000 并且是 100 的倍数	3	1000	取款成功
		略小于 3000	4	2999	取款金额为 100 的倍数
		等于 3000	5	3000	取款成功

5. 某程序要求输入日期，规定变量 month、day、year 的取值范围为：1≤month≤12，1≤day≤31，1958≤year≤2058，试用边界值分析法设计测试用例。

答：本例中要求输入三个变量的值，按照标准边界值分析测试，该程序会产生 4n+1=13 个测试用例，见下表。

标准边界值测试用例表

测试方法	测试区间	测试用例编号	预期输入			预期输出
			year	month	day	
标准界值分析法	year 为最小值 month 为正常值 day 为正常值	1	1958	6	10	有效
	year 略大于最小值 month 为正常值 day 为正常值	2	1959	6	10	有效
	year 略小于最大值 month 为正常值 day 为正常值	3	2057	6	10	有效
	year 等于最大值 month 为正常值 day 为正常值	4	2058	6	10	有效

续表

测试方法	测试区间	测试用例编号	预 期 输 入			预期输出
			year	month	day	
标准界值分析法	year 为正常值 month 为最小值 day 为正常值	5	2013	1	10	有效
	year 为正常值 month 略大于最小值 day 为正常值	6	2013	2	10	有效
	year 为正常值 month 略小于最大值 day 为正常值	7	2013	11	10	有效
	year 为正常值 month 等于最大值 day 为正常值	8	2013	12	10	有效
	year 为正常值 month 为正常值 day 为最小值	9	2013	6	1	有效
	year 为正常值 month 为正常值 day 略大于最小值	10	2013	6	2	有效
	year 为正常值 month 为正常值 day 略小于最大值	11	2013	6	30	有效
	year 为正常值 month 为正常值 day 等于最大值	12	2013	6	31	无效
	year 为正常值 month 为正常值 day 为正常值	13	2013	6	10	有效

按照健壮边界值分析测试程序会产生 6n+1=19 测试用例，见下表。

健壮边界值测试用例表

测试方法	测试区间	测试用例编号	预 期 输 入			预期输出
			year	month	day	
健壮边界值分析法	year 为最小值 month 为正常值 day 为正常值	1	1958	6	10	有效
	year 略大于最小值 month 为正常值 day 为正常值	2	1959	6	10	有效

续表

测试方法	测试区间	测试用例编号	预期输入			预期输出
			year	month	day	
健壮边界值分析法	year 略小于最大值 month 为正常值 day 为正常值	3	2057	6	10	有效
	year 等于最大值 month 为正常值 day 为正常值	4	2058	6	10	有效
	year 为正常值 month 为最小值 day 为正常值	5	2013	1	10	有效
	year 为正常值 month 略大于最小值 day 为正常值	6	2013	2	10	有效
	year 为正常值 month 略小于最大值 day 为正常值	7	2013	11	10	有效
	year 为正常值 month 等于最大值 day 为正常值	8	2013	12	10	有效
	year 为正常值 month 为正常值 day 为最小值	9	2013	6	1	有效
	year 为正常值 month 为正常值 day 略大于最小值	10	2013	6	2	有效
	year 为正常值 month 为正常值 day 略小于最大值	11	2013	6	30	有效
	year 为正常值 month 为正常值 day 等于最大值	12	2013	6	31	无效
	year 为正常值 month 为正常值 day 为正常值	13	2013	6	10	有效
	year 略小于最小值 month 为正常值 day 为正常值	14	1957	6	10	无效
	year 略大于最大值 month 为正常值 day 为正常值	15	2059	6	10	无效

续表

测试方法	测试区间	测试用例编号	预期输入			预期输出
			year	month	day	
健壮边界值分析法	year 为正常值 month 略小于最小值 day 为正常值	16	2013	0	10	无效
	year 为正常值 month 略大于最大值 day 为正常值	17	2013	13	10	无效
	year 为正常值 month 为正常值 day 略小于最小值	18	2013	6	0	无效
	year 为正常值 month 为正常值 day 略大于最大值	19	2013	6	32	无效

6. A　7. D　8. C　9. B　10. B　11. B　12. D　13. D　14. C
15. B　16. D　17. D

习题 4 答案

1. 简述白盒测试用例的设计方法，并进行分析总结。

答：白盒测试用例设计方法主要有逻辑覆盖和独立路径测试。

从覆盖源程序语句的详尽程度分析，逻辑覆盖主要有以下不同的覆盖标准：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。实际项目中，由于程序内部的逻辑存在不确定性和无穷性，尤其对于大规模复杂软件，不必采用所有的覆盖指标，而应根据实际情况选择合适的覆盖指标。

独立路径测试是在程序控制流图的基础上，通过分析控制结构的环路复杂性，导出可执行的独立路径集合，从而设计出相应的测试用例。设计出的测试用例要保证被测程序的每条可执行的独立路径至少被执行一次。独立路径测试给出了满足路径覆盖指标所需测试用例的下限，同时给出了语句覆盖的上限，它可以确保对所有相互独立的决策结果进行测试。

2. 分析归纳逻辑覆盖的各种策略，并比较每种覆盖的特点，分析在怎样的情况下采用何种覆盖方式。

答：语句覆盖是选择足够多的测试数据，使被测程序中每个语句至少执行一次。语句覆盖是最弱的逻辑覆盖标准。

判定覆盖又叫分支覆盖，它不仅每个语句必须至少执行一次，而且每个判定表达式的每种可能的结果都应该至少执行一次。判定条件覆盖比语句覆盖强，但是对程序逻辑的覆盖程度仍然不高。

条件覆盖的含义是，使判定表达式中的每个条件都取到各种可能的结果。条件覆盖通常比判定覆盖强，但是也可能有相反的情况：虽然每个条件都取到了两个不同的结果，判定表达式却始终只取一个值。

判定/条件覆盖的含义是，选取足够多的测试数据，使得判定表达式中的每个条件都取到各种可能的值，而且每个判定表达式也都取到各种可能的结果。但有时判定/条件覆盖也并不比条件覆盖更强。

条件组合覆盖是更强的逻辑覆盖标准，它要求选取足够的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。满足条件组合覆盖标准的测试数据，也一定满足判定覆盖、条件覆盖和判定/条件覆盖标准。因此，条件组合覆盖是前述几种覆盖标准中最强的。但是，满足条件组合覆盖标准的测试数据并不一定能使程序中的每一条路径都执行到。

路径覆盖的定义是选取足够多的测试数据，使程序的每一条可能路径都至少执行一次。但在实际问题中，一个不太复杂的程序，其路径数都可能是一个庞大的数字，以致要在测试中覆盖所有的路径是不可能实现的。即使对于路径数有限的程序做到了路径覆盖，也不能保证被测程序的正确性。

3. 请针对以下代码按照各种覆盖方法设计测试用例。

```
if (a>5&& b==3 && (c>2 || d<7))
{
    Statement1;
}else{
    Statement2;
}
```

答：在本例中

条件 4 个：a>5 记为 T_1，a<=5 记为 F_1；
b==3 记为 T_2，b!=3 记为 F_2；
c>2 记为 T_3，c<=2 记为 F_3；
d<7 记为 T_4，d>=7 记为 F_4.

判定 1 个：a>5&& b==3 && (c>2 || d<7)为真时记为 T_{D1}；判定为假，记为 F_{D1}。

语句 2 条：Statement1 和 Statement2

路径 2 条：R1　当判定为真时　R2　当判定为假时

(1) 语句覆盖测试用例设计

逻辑覆盖类型	语句	条件	判定	测试用例编号	预期输入			
					a	b	c	d
语句覆盖	S_1	$T_1T_2T_3F_4$	T_{D1}	1	8	3	3	8
	S_2	$F_1T_2T_3T_4$	F_{D1}	2	4	3	3	6

(2) 判定覆盖测试用例设计

逻辑覆盖类型	语句	条件	判定	测试用例编号	预期输入			
					a	b	c	d
判定覆盖	S_1	$T_1T_2T_3F_4$	T_{D1}	1	8	3	3	8
	S_2	$F_1T_2T_3T_4$	F_{D1}	2	4	3	3	6

(3) 条件覆盖测试用例设计

逻辑覆盖类型	语句	条件	判定	测试用例编号	预期输入			
					a	b	c	d
条件覆盖	S_1	$T_1F_2T_3F_4$	F_{D1}	1	8	2	3	8
	S_2	$F_1T_2F_3T_4$	F_{D1}	2	4	3	2	6

(4) 判定条件覆盖测试用例设计

逻辑覆盖类型	语句	条件	判定	测试用例编号	预期输入			
					a	b	c	d
判定条件覆盖	S_1	$T_1T_2T_3T_4$	T_{D1}	1	8	3	3	8
	S_2	$F_1F_2F_3F_4$	F_{D1}	2	4	2	2	6

(5) 条件组合覆盖测试用例设计

条件组合覆盖要求每个判定中的所有可能的原子条件取值组合至少执行一次，这里只有一个判定，四个条件组合，每个条件只能为 true 或者 false，所以本例中条件组合覆盖测试用例数为 $2^4=16$ 个。

逻辑覆盖类型	语句	条件	判定	测试用例编号	预期输入			
					a	b	c	d
条件组合覆盖	S_1	$T_1T_2T_3T_4$	T_{D1}	1	6	3	3	6
	S_2	$F_1T_2T_3T_4$	F_{D1}	2	4	3	3	6
	S_2	$T_1F_2T_3T_4$	F_{D1}	3	6	2	3	6
	S_1	$T_1T_2F_3T_4$	T_{D1}	4	6	3	1	6
	S_1	$T_1T_2T_3F_4$	T_{D1}	5	6	3	3	7
	S_2	$F_1F_2T_3T_4$	F_{D1}	6	4	2	3	6
	S_2	$F_1T_2F_3T_4$	F_{D1}	7	4	3	1	6
	S_2	$F_1T_2T_3F_4$	F_{D1}	8	4	3	3	8
	S_2	$T_1F_2F_3T_4$	F_{D1}	9	6	2	2	6
	S_2	$T_1F_2T_3F_4$	F_{D1}	10	6	2	3	8
	S_2	$T_1T_2F_3F_4$	F_{D1}	11	6	3	1	8
	S_2	$F_1F_2F_3T_4$	F_{D1}	12	4	2	1	6
	S_2	$T_1F_2F_3F_4$	F_{D1}	13	6	2	1	8
	S_2	$F_1T_2F_3F_4$	F_{D1}	14	4	3	1	8
	S_2	$F_1F_2T_3F_4$	F_{D1}	15	4	2	3	8
	S_2	$F_1F_2F_3F_4$	F_{D1}	16	4	2	1	8

（6）路径覆盖测试用例设计

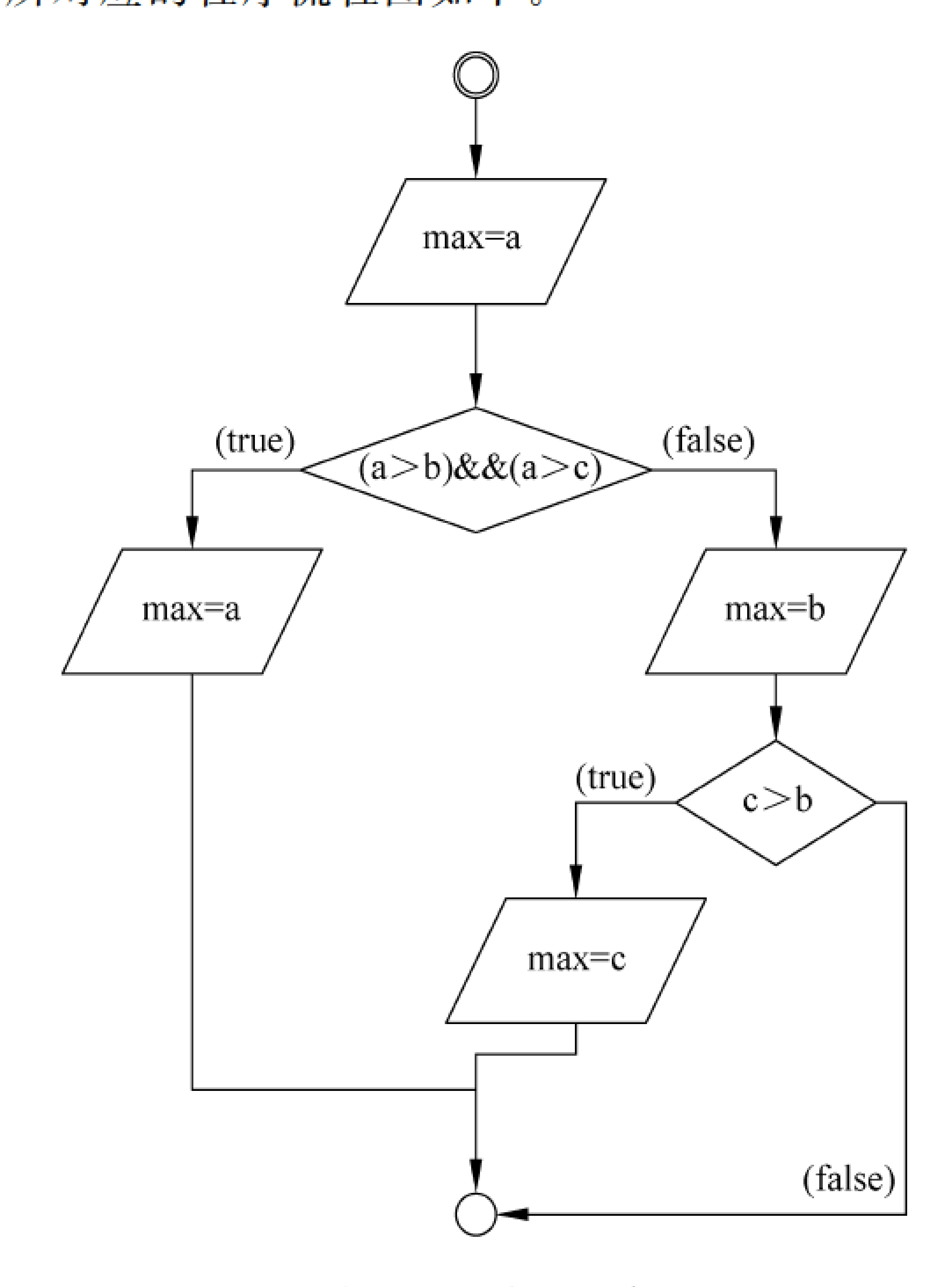
逻辑覆盖类型	语句	条件	判定	测试用例编号	预期输入			
					a	b	c	d
路径覆盖	S_1	$T_1T_2T_3F_4$	T_{D1}	1	8	3	3	8
	S_2	$F_1T_2T_3T_4$	F_{D1}	2	4	3	3	6

（注：测试用例设计数据不唯一，读者可自由掌握。）

4. 对以下程序设计测试用例，画出程序流程图。使用独立路径测试法，进行测试用例的设计，并给出 getMax 方法的返回值。

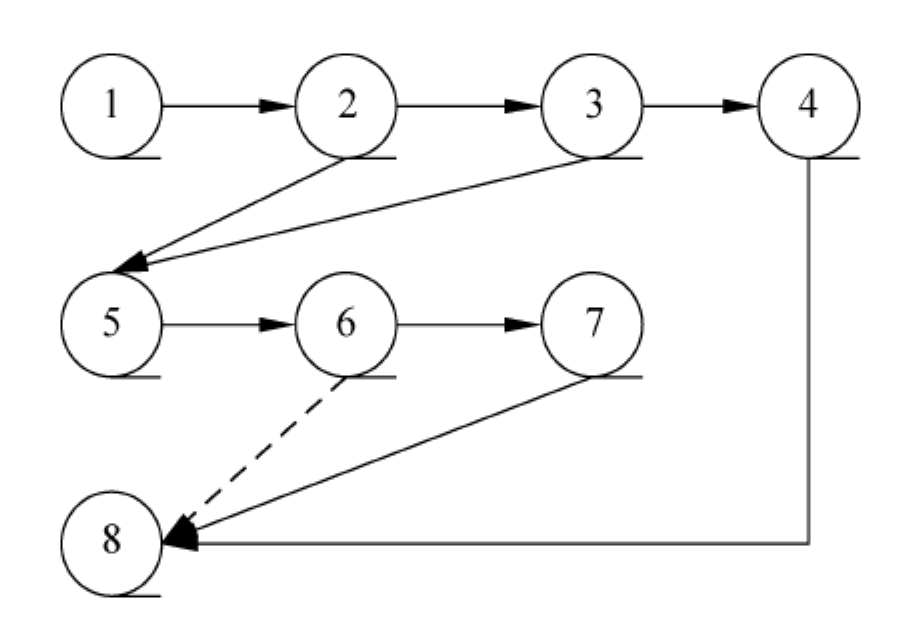
```
public intgetMax(inta,intb,intc)
{
    intmax = a;
    if((a > b)&&(a > c))
    {
        max = a;
    }else{
        max = b;
        if(c > b){
            max = c;
        }
    }
    return max;
}
```

答：getMax 方法所对应的程序流程图如下。



getMax 方法所对应的程序流程图

getMax 方法所对应的控制流图如下。



getMax 方法所对应的控制流图

节点 1：max＝a；

节点 2：a＞b；

节点 3：a＞c；

节点 4：max＝a；

节点 5：max＝b；

节点 6：c＞b；

节点 7：max＝c；

节点 8：return max。

环形复杂度：由控制流图能得到环形复杂度 V(G)＝E－N＋2＝10－8＋2＝4(E 是流图中边的数量，N 是流图中节点的数量)。

独立路径个数为 4，对应的路径分别为：

① 1→2→5→6→8

② 1→2→3→5→6→8

③ 1→2→5→6→7→8

④ 1→2→3→4→8

根据 4 条独立路径，设计测试用例如下表所示。

案例 1 独立路径测试法测试用例

白盒测试类型	测试用例编号	预 期 输 入			覆盖路径	预期输出
		a	b	c		
独立路径测试	1	2	3	1	①	max＝3
	2	?	?	?	②	?
	3	2	3	4	③	max＝4
	4	4	2	3	④	max＝4

5. 面向对象的白盒测试和传统的白盒测试有什么区别?

面向对象的白盒测试把类作为一个单元来进行测试。测试分为两层：第一层考虑类中各独立方法的代码；第二层考虑方法之间的相互作用。

(1) 独立方法的白盒测试

面向对象的白盒测试第一层是考虑各独立的方法，这可以与过程的测试采用同样的方

法，两者之间最大的差别在于方法改变了它所在实例的状态，这就要取得隐藏的状态信息来估算测试的结果，传给其他对象的消息被忽略，而以桩来代替，并根据所传的消息返回相应的值，测试数据要求能完全覆盖类中代码，可以用传统的测试技术来获取。

(2) 方法之间的测试

第二层要考虑一个方法调用本对象类中的其他方法和从一个类向其他类发送信息的情况。单独测试一个方法时，只考虑其本身执行的情况，而没有考虑动作的顺序问题，测试用例中加入了激发这些调用的信息，以检查它们是否正确运行了。对于同一类中方法之间的调用，一般只需要极少甚至不用附加数据，因为方法都是对类进行存取，故这一类测试的准则是要求遍历类的所有主要状态。

6. D　7. A　8. D　9. C　10. D　11. A　12. B　13. D　14. D
15. C　16. A　17. A　18. B

习题 5 答案

1. 软件测试管理的内容包括哪些？

答：具体的测试管理内容如下：

(1) 测试方案管理：单元测试、集成测试和产品测试的测试计划的录入、修改、删除、查询和打印。

(2) 测试案例管理：测试案例的增、删、改、复制和查询；测试案例测试情况的管理，如测试状态包括：未测试、测试中、已测试；测试结果分为：通过、未实现、存在问题等；测试案例输入、编号和归档。

(3) 测试流程管理：测试进度管理、测试流程标识、测试日志及状态报告。

(4) 问题报告管理：问题报告处理流程(问题报告→整改报告)、实现问题报告与测试案例的关联。

(5) 测试报告管理：生成单元测试、集成测试和产品测试的测试报告。

2. 简要阐述软件缺陷所包括的内容。

答：软件测试的目的是为了尽早发现软件系统中的缺陷，软件测试过程简单说就是围绕缺陷进行的、对缺陷的跟踪管理。

软件缺陷属性包括缺陷标识、缺陷类型、缺陷严重程度、缺陷产生可能性、缺陷优先级、缺陷状态、缺陷起源、缺陷来源、缺陷原因。

缺陷标识：是标记某个缺陷的唯一的表示，可以使用数字序号表示。

缺陷类型：是根据缺陷的自然属性划分缺陷种类。包括功能、用户界面、文档、软件包、性能、系统/模块接口等。

缺陷严重程度：是指因缺陷引起的故障对软件产品的影响程度，所谓“严重性”指的是在测试条件下，一个错误在系统中的绝对影响。分为致命、严重、一般、较小缺陷。

缺陷产生的可能性：指缺陷在产品中发生的可能性，通常可以用频率来表示。

缺陷优先级：指缺陷必须被修复的紧急程度。“优先级”的衡量抓住了在严重性中没有考虑的重要程度因素。

缺陷状态：指缺陷通过一个跟踪修复过程的进展情况，也就是在软件生命周期中的状态基本定义。

缺陷起源：缺陷引起的故障或事件第一次被检测到的阶段。

缺陷来源：指缺陷所在的地方，如文档、代码等。

缺陷根源：指造成上述错误的根本因素，以寻求软件开发流程的改进、管理水平的提高。

3. 软件测试自动化的优缺点是什么？

答：软件测试自动化的优点如下。

(1) 可以提高测试效率，使测试人员更加专注于新的测试模块的建立和开发，从而提高测试覆盖率；

(2) 测试自动化使测试资产的管理数字化，并使测试资产在整个软件测试生命周期内得以复用，这个特点在功能测试和回归测试中尤其具有意义；

(3) 通过测试流程的自动化管理提高了测试过程的有效性。

软件测试自动化的缺点如下。

(1) 自动化测试不能取代手工测试来完成所有的测试任务。如果测试只是偶尔执行，或者待测系统经常变动、不稳定，测试需要大量的人工参与时，就不适宜采用自动化测试。

(2) 自动测试工具本身不具有想象力，只是按运行机制执行。而手工测试时测试执行者可以在测试中判断测试输出是否正确，以及改进测试，还可以处理意外事件。如网络连接中断，此时必须重新建立连接。手工测试时可以及时处理该意外，而自动化测试时该意外事件一般都会导致测试的中止。

(3) 对测试质量的依赖性较大。进行自动化测试实际上仅仅意味着测试的结果与期望值相同，因此测试的有效性很大程度上依赖于自动化测试本身的质量。确保测试的质量往往比自动化测试更为重要，所以要投入精力对测试软件进行必要的检测。

(4) 自动化测试成本可能高于手工测试。自动化测试的成本大致由以下几个部分组成：自动测试开发成本、运行成本、测试维护成本、工具成本和其他相关任务成本。

(5) 自动化测试可能会制约软件开发。应用软件的变化对自动化测试的影响要比手工测试更大一些，软件的部分改变有可能使自动化测试崩溃。而设计和实施自动化测试要比手工测试开销大，并需要维护，所以，对自动化测试影响较大的软件修改可能受到限制。

4. D　5. D　6. A　7. D　8. B　9. B　10. A　11. D　12. D

13. A　14. B　15. A　16. A

习题 6 答案

1. QTP 的工作流程包括哪 5 个步骤？

答：(1) 录制测试脚本。例如单击链接或图像，或者提交数据表单。

(2) 插入检查点。检查页面、对象或文本字符串中的特定值或属性。

(3) 测试脚本参数化。测试网站或应用程序时，可以参数化测试脚本以检查应用程序如何使用不同数据执行相同的操作。

(4) 运行测试脚本。测试脚本将从其第一行开始运行直至测试结束时停止。在运行中，QTP 将连接到待测网站或应用程序，执行测试脚本中的每一项操作，检查所有指定的文

本字符串、对象或表。如果使用数据表参数对测试进行了参数化，测试工具还将对定义的每组数据值重复该测试。

(5) 测试结果分析。测试人员可以查看结果的概要，也可以查看详细报告，进行测试报告分析。

2. 检查点的类型有哪些？

答：(1) 标准检查点，检查对象的属性。

(2) 图片检查点，检查图片的属性，例如检查图片的来源文件是否正确。

(3) 表格检查点，检查表格的内容，例如检查表格的内容是否正确。

(4) 网页检查点，检查网页的属性，例如检查网页加载的时间或网页是否含有不正确的链接。

(5) 文字/文字区域检查点，检查网页或窗口上出现的文字是否正确，例如检查登录系统后是否出现登录成功的文字。

(6) 图像检查点，提取网页和窗口的画面检查画面是否正确，例如检查网页或者网页的一部分是否如期显示。

(7) 数据库检查点，检查数据库的内容是否正确，例如检查数据库查询的值是否正确。

(8) XML 检查点，检查 XML 文件的内容。

3. 测试数据参数化的优势是什么？

答：使用 QTP 可以通过将固定值替换为参数，扩展基本测试或组件的范围。该过程(称为参数化)大大提高了测试或组件的功能和灵活性。

可在 QTP 中使用参数功能，通过参数化测试或组件所使用的值来增强测试或组件。参数是一种从外部数据源或生成器赋值的变量。

QTP 可以参数化测试或组件中的步骤和检查点中的值。还可以参数化操作参数的值。如果希望参数化测试或组件多个步骤中的同一个值，可能需要考虑使用数据驱动器，而不是手动添加参数。

4. 略

习题 7 答案

1. 简述负载场景如何设置，主要包括哪些核心步骤。

答：使用 LoadRunner 进行负载测试通常由 5 个阶段组成：制订测试计划，创建测试脚本，定义场景，执行场景和结果分析。

(1) 制订测试计划。定义性能测试要求，例如并发用户的数量、典型业务流程和所需响应时间。

(2) 创建测试脚本。将最终用户的活动捕获到脚本中。

(3) 定义场景。使用 Controller 设置负载测试环境。

(4) 执行场景。通过 Controller 驱动、管理和监控负载测试。

(5) 结果分析。使用 Analysis 创建图和报告并评估性能。

2. 举例说明负载测试目标包括哪几项。

负载测试目标即负载测试是否有效的一个衡量标准。通常情况下需要确立五种不同类型的目标：并发 Vuser 数、每秒点击次数、每秒事务数、每分钟页面数和场景的事务响应时间。

当想了解可运行各种业务流程的 Vuser 总数，则可以使用虚拟用户目标类型。

当想知道服务器的稳定性，则可以使用每秒点击次数、每分钟页面数或每秒事务数目标类型。

当想知道所需的完成事务的响应时间，则可以使用事务响应时间目标类型。如：用户希望在 5s 内得到服务器的反馈信息。

这五个目标也有一定的关联和制约关系。如：Vuser 数不断增加的话，必然会引起事务响应时间的增加。在目标的确定过程中，需要测试人员的经验。对待测服务器的处理能力以及性能有一定的初步判断后，确定负载测试目标。

3. 略

4. 略

习题 8 答案

1. TestCase 和 TestSuite 有什么区别？

答：TestCase(测试用例)：包含很多以 test 开头的方法，用来测试被测类中的 public 类型的方法。通过比较方法的输出结果和预期结果是否相同，来判断本次测试是否成功或者失败。

TestSuite (测试套件)：TestCase 并不能孤立地使用，它总是需要依附在 TestSuite 中。TestSuite 代表一个或者一组 TestCase。通过 TestSuite 把 TestCase 很好地组合在一起，形成一组测试单元，也称为测试套件。

2. 断言方法 assertEquals()是如何进行测试的？

答：assertEquals(Object expected,Object actual)：判断第一个参数和第二个参数是否相等，如果不相等时，提示错误信息。第一个参数是期望对象，第二个参数是实际对象。

3. A　　4. A

5. 按题目要求编写相关类，并使用 JUnit 对计算周长和面积的方法进行单元测试。

梯形 Lader 类具有类型为 double 的上底、下底、高、面积属性，具有返回面积的功能，包括一个构造方法对上底、下底、高进行初始化。圆形 Circle 类具有类型为 double 的半径、周长和面积属性，具有返回周长、面积的功能，包括一个构造方法对半径进行初始化。

答：

```
public class Lader
{
    double a, b, h;
    double area;
    double getArea()
    {
        area = (a + b) * h/2;
        return area;
    }


    Lader(double a, double b, double h) {
        this.a = a;
        this.b = b;
```

```
        this.h = h;
    }
}

public class Circle
{
    double r;
    double len, area;
    double getLen()
    {
        len = 2 * 3.14 * r;
        return len;
    }
    double getArea()
    {
        area = 3.14 * r * r;
        return area;
    }

    Circle(double r) {
        this.r = r;
    }
}

import static org.junit.Assert.*;
import junit.framework.TestCase;
import org.junit.Test;
public class CircleTest extends TestCase {
    Circle circle = new Circle(5);
    @Test
    public void testGetLen() {
        assertEquals(31.4, circle.getLen());        //预期圆形周长结果为 31.4
    }

    @Test
    public void testGetArea() {
        assertEquals(78.5, circle.getArea());       //预期圆形面积结果为 78.5
    }
}
import static org.junit.Assert.*;
import junit.framework.TestCase;
import org.junit.Test;
public class LaderTest extends TestCase {
    Laderlader = new Lader(4,5,10);
    @Test
    public void testGetArea() {
        assertEquals(45.0, lader.getArea());        //预期梯形面积为 45
    }
}
```

软件测试国家标准

测试计划（GB 8567—1988）

1　引言

1.1　编写目的

本测试计划的具体编写目的，指出预期的读者范围。

1.2　背景

说明：

a. 测试计划所从属的软件系统的名称；

b. 该开发项目的历史，列出用户和执行此项目测试的计算中心，说明在开始执行本测试计划之前必须完成的各项工作。

1.3　定义

列出本文件中用到的专门术语的定义和外文首字母组词的原词组。

1.4　参考资料

列出要用到的参考资料，如：

a. 本项目的经核准的计划任务书或合同、上级机关的批文；

b. 属于本项目的其他已发表的文件；

c. 本文件中各处引用的文件、资料，包括所要用到的软件开发标准，列出这些文件的标题、文件编号、发表日期和出版单位，说明能够得到这些文件资料的来源。

2　计划

2.1　软件说明

提供一份图表，并逐项说明被测软件的功能、输入和输出等质量指标，作为叙述测试计划的提纲。

2.2　测试内容

列出组装测试和确认测试中的每一项测试内容的名称标识符、这些测试的进度安排以及这些测试的内容和目的，例如模块功能测试、接口正确性测试、数据文卷存取的测试、运行时间的测试、设计约束和极限的测试等。

2.3　测试 1（标识符）

给出这项测试内容的参与单位及被测试的部位。

2.3.1　进度安排

给出对这项测试的进度安排，包括进行测试的日期和工作内容（如熟悉环境，培训，准备输入数据等）。

2.3.2　条件

陈述本项测试工作对资源的要求，包括：

a. 设备所用到的设备类型、数量和预定使用时间；

b. 软件列出将被用来支持本项测试过程而本身又并不是被测软件的组成部分的软件，

如测试驱动程序、测试监控程序、仿真程序、桩模块等；

c. 人员列出在测试工作期间预期可由用户和开发任务组提供的工作人员的人数、技术水平及有关的预备知识，包括一些特殊要求，如倒班操作和数据录入人员。

2.3.3　测试资料

列出本项测试所需的资料，如：

a. 有关本项任务的文件；

b. 被测试程序及其所在的媒体；

c. 测试的输入和输出举例；

d. 有关控制此项测试的方法、过程的图表。

2.3.4　测试培训

说明或引用资料说明为被测软件的使用提供培训的计划。规定培训的内容、受训的人员及从事培训的工作人员。

2.4　测试2(标识符)

用与本测试计划2.3条相类似的方式说明用于另一项及其后各项测试内容的测试工作计划。

3　测试设计说明

3.1　测试1(标识符)

说明对第一项测试内容的测试设计考虑。

3.1.1　控制

说明本测试的控制方式，如输入是人工、半自动或自动引入、控制操作的顺序以及结果的记录方法。

3.1.2　输入

说明本项测试中所使用的输入数据及选择这些输入数据的策略。

3.1.3　输出

说明预期的输出数据，如测试结果及可能产生的中间结果或运行信息。

3.1.4　过程

说明完成此项测试的一个个步骤和控制命令，包括测试的准备、初始化、中间步骤和运行结束方式。

3.2　测试2(标识符)

用与本测试计划3.1条相类似的方式说明第2项及其后各项测试工作的设计考虑。

4　评价准则

4.1　范围

说明所选择的测试用例能够接查的范围及其局限性。

4.2　数据整理

陈述为了把测试数据加工成便于评价的适当形式，使得测试结果可以同已知结果进行比较而要用到的转换处理技术，如手工方式或自动方式；如果是用自动方式整理数据，还要说明为进行处理而要用到的硬件、软件资源。

4.3　尺度

说明用来判断测试工作是否能通过的评价尺度，如合理的输出结果的类型、测试输出结果与预期输出之间的容许偏离范围、允许中断或停机的最大次数。

测试分析报告(GB 8567—1988)

1　引言

1.1　编写目的

说明这份测试分析报告的具体编写目的,指出预期的阅读范围。

1.2　背景

说明:

a. 被测试软件系统的名称;

b. 该软件的任务提出者、开发者、用户及安装此软件的计算中心,指出测试环境与实际运行环境之间可能存在的差异以及这些差异对测试结果的影响。

1.3　定义

列出本文件中用到的专问术语的定义和外文首字母组词的原词组。

1.4　参考资料

列出要用到的参考资料,如:

a. 本项目的经核准的计划任务书或合同、上级机关的批文;

b. 属于本项目的其他已发表的文件;

c. 本文件中各处引用的文件、资料,包括所要用到的软件开发标准,列出这些文件的标题、文件编号、发表日期和出版单位,说明能够得到这些文件资料的来源。

2　测试概要

用表格的形式列出每一项测试的标识符及其测试内容,并指明实际进行的测试工作内容与测试计划中预先设计的内容之间的差别,说明作出这种改变的原因。

3　测试结果及发现

3.1　测试 1(标识符)

把本项测试中实际得到的动态输出(包括内部生成数据输出)结果同对于动态输出的要求进行比较,陈述其中的各项发现。

3.2　测试 2(标识符)

用类似本报告 3.1 条的方式给出第 2 项及其后各项测试内容的测试结果和发现。

4　对软件功能的结论

4.1　功能 1(标识符)

4.1.1　能力

简述该项功能,说明为满足此项功能而设计的软件能力以及经过一项或多项测试已证实的能力。

4.1.2　限制

说明测试数据值的范围(包括动态数据和静态数据),列出就这项功能而言,测试期间在该软件中查出的缺陷、局限性。

4.2　功能 2(标识符)

用类似本报告 4.1 的方式给出第 2 项及其后各项功能的测试结论。

……

5　分析摘要

5.1　能力

陈述经测试证实了的本软件的能力。如果所进行的测试是为了验证一项或几项特定性能要求的实现,应提供这方面的测试结果与要求之间的比较,并确定测试环境与实际运行环境之间可能存在的差异对能力的测试所带来的影响。

5.2　缺陷和限制

陈述经测试证实的软件缺陷和限制,说明每项缺陷和限制对软件性能的影响,并说明全部测得的性能缺陷的累积影响和总影响。

5.3　建议

对每项缺陷提出改进建议,如:

a. 各项修改可采用的修改方法;

b. 各项修改的紧迫程度;

c. 各项修改预计的工作量;

d. 各项修改的负责人。

5.4　评价

说明该项软件的开发是否已达到预定目标,能否交付使用。

6　测试资源消耗

总结测试工作的资源消耗数据,如工作人员的水平级别数量、使用机器时间消耗等。